UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

Desarrollo de una aplicación Android de una red social orientada a la televisión

Hugo Cortijo Navascués
Tutor: Gonzalo Martínez

Julio 2014

RESUMEN

La idea de desarrollar una aplicación *Android* de tipo *red social* orientada a la televisión nace a partir de varios motivos.

Uno de ellos es el gran uso de las aplicaciones de nuestro teléfono móvil mientras vemos la televisión, un fenómeno que en la industria televisiva se denomina "segunda pantalla" haciendo referencia al dispositivo, móvil o tableta, que un espectador utiliza a la vez que está viendo cualquier programa. De acuerdo con los últimos estudios de mercado, un 65% de los telespectadores españoles utilizan su *smartphone* mientras ve la televisión. Entre las aplicaciones más utilizadas destaca *Twitter*, pues a los usuarios de esta red social les gusta comentar qué está sucediendo en la televisión, y se estima que en España un tercio de tweets enviados en horario prime time son relacionados con algún programa de televisión, especialmente partidos de futbol, reality shows o series de ficción. Adicionalmente, un 45% de los españoles chatean por *Whatsapp* mientras ven la televisión, un 40% buscan información en Internet sobre el contenido que están viendo y un 25% indagan en la red sobre productos que ven en un programa o en anuncio [1].

Todo esto sería interesante hacerlo a través de una única aplicación. Esto es justamente lo que se pretende en este proyecto, una aplicación móvil destinada a mejorar la experiencia de todas las personas a las que les gustan estos nuevos hábitos de descubrir y participar, y con la que se quiere fomentar una nueva forma de ver la televisión, más divertida, social, participativa y personalizada.

En este documento se plasma cómo surge Vuqio y cuál es su objetivo. Una vez definido estos aspectos previos se concretan las necesidades del usuario para poder desarrollar una solución. Dicha solución consta de un diseño de la interfaz gráfica detallando la resolución de los requisitos, diseños preliminares y finales de la base de datos para gestionar los chats y alarmas del usuario y aspectos más técnicos como son la arquitectura de la aplicación y su seguridad. Tras definir la aplicación se explicará la solución tomada en partes concretas de la aplicación y qué técnicas y/o librerías se han utilizado y por qué, dando una explicación detallada de los motivos.

Se garantizará el correcto funcionamiento de la aplicación gracias a la batería de pruebas expuestas en este documento.

La aplicación se ha desarrollado en Buqio SL una startup creada por Pablo Vaquero y José Cantera. Comencé el proyecto en periodo de prácticas externas por la Universidad Autónoma de Madrid por un periodo de 3 meses y después como trabajo durante más de un año. El desarrollo ha sido supervisado por Pablo Vaquero Co-fundador de Buqio SL y Miguel Angel Diaz responsable de desarrollo en Buqio SL.

Este proyecto está disponible en *Google Play Store* y cuenta con más de 1000 descargas.

INDICE

RESUMEN ..... **I**

INDICE ..... **III**

GLOSARIO ..... **V**

INTRODUCCIÓN **1**
1.1 MOTIVACIÓN Y DEFINICIÓN DEL PROBLEMA 1
1.2 OBJETIVOS DEL PROYECTO 1

2. ESTUDIO DEL ESTADO DEL ARTE **3**
2.1 APLICACIONES RELACIONADAS 3
TOCKIT *3*
ZEEBOX (BEAMLY) *4*

3. DEFINICIÓN DEL PROYECTO **7**
3.1 REQUERIMIENTOS FUNCIONALES 7
3.2 REQUERIMIENTOS NO FUNCIONALES 11

4. DISEÑO DE LA SOLUCIÓN **13**
4.1 DISEÑO DE LA INTERFAZ 13
4.1.1 Interfaz principal: Guía de Televisión y Chat Global. *17*
4.1.2 Interfaz información detallada del programa *20*
4.1.3 Interfaz menú lateral *24*
4.1.3.1 Perfil del usuario *24*
4.1.3.2 Gestión de alarmas *24*
4.1.3.3 Gestión de conversaciones abiertas *26*
4.1.4 Interfaz notificaciones *26*
4.2 DISEÑO DE LA BASE DE DATOS 28
4.2.1 Chat *28*
4.2.1.1 Diseño Preliminar *28*
4.2.1.2 Diseño Final *28*
4.2.2 Alarmas *29*
4.2.2.1 Diseño Preliminar *29*
4.2.2.2 Diseño Final *30*
4.3 ARQUITECTURA DEL SISTEMA 30
4.4 TECNOLOGÍAS UTILIZADAS 33
4.4.1 ANDROID SDK *33*
4.4.2 ECLIPSE *33*
4.5 SEGURIDAD 33

5. SOLUCIÓN **35**
5.1 ELEMENTOS BÁSICOS 35
5.2 LIBRERÍAS 36
5.2.1 Red Social *36*
5.2.2 Peticiones Asíncronas *37*
5.2.3 Interfaz *39*
5.2.4 Análisis de errores *41*
5.2.5 Conexión *42*
5.2.6 Notificaciones *42*
5.3 INTERFAZ 43
5.3.1 Guía de televisión *44*
5.3.2 Información detallada del programa *45*
5.4 TÉCNICAS DE DESARROLLO 47

5.4.1 Técnica ViewHolder *47*
5.4.2 Triggers *48*
6. PRUEBAS **49**
6.1 PRUEBAS DE USABILIDAD 49
6.2 PRUEBAS DE SISTEMA 50
7. ANÁLISIS **51**
8. CONCLUSIONES **53**

GLOSARIO

Twitter	Red social.
Facebook	Red social
Google Play Store	Servicio de tienda virtual que permite distribuir aplicaciones para el sistema Android.
gamification	Uso de mecánicas de juegos fuera del contexto de los mismos.
Notificaciones Push	Servicio proporcionado por Google Cloud Messaging para controlar el envío de notificaciones a un dispositivo móvil.
Backend	Servidores que proporcionan los servicios.
URL	Unifor Resource Locator. Dirección única de un recurso en Internet.
Smartphones	Teléfono móvil capaz de realizar actividades semejantes a una computadora.
Tablet	Dispositivo con mayores dimensiones que un Smartphone.
Websocket	Tecnología que permite la comunicación bidireccional sobre un socket TCP.
XML	Extensible Markup Languaje. Lenguaje de etiquetado para el intercambio de datos.
JSON	Javascript Object Notation. Formato ligero de intercambio de datos.
Ruby On Rails	Framework escrito en ruby siguiendo la arquitectura MVC.
MVC	Patrón de diseño software que separa los datos y lógica de la interfaz del usuario.
API REST	Es un tipo de arquitectura de desarrollo web para crear APIs para servicios orientados a Internet.
Redis	Caché de datos con estructura.
NoSQL	Base de datos no relacionales.
MongoDB	Base de datos no relacional que permite indexar campos para mejorar el tiempo de consulta.
GitHub	Sistema de control de versiones.

INTRODUCCIÓN

1.1 Motivación y definición del problema

Debido al aumento de *smartphones* y aplicaciones que utilizamos mientras vemos la televisión surge la necesidad de unir todas ellas en una única aplicación capaz de mejorar la experiencia de todas aquellas personas que quieren descubrir otra forma de ver la televisión, más social, divertida y personalizada.

Así nace *Vuqio,* ya disponible en el ***Google Play Store***, una aplicación que requiere mostrar toda la programación televisiva actual y de días posteriores donde el usuario podrá comunicarse con otros usuarios a través de un chat, compartir contenidos en sus cuentas de *Facebook* y Twitter, estar informado del comienzo de sus programas favoritos mediante un gestor de alarmas y estar informado de lo que se está comentando en *Twitter* y en el programa en directo.

Vuqio es una plataforma de televisión social, que permite al usuario tener una nueva experiencia al ver la televisión mediante la conexión de los usuarios por sus programas favoritos. Los aspectos fundamentales de Vuqio son:

- Divertido: Para la audiencia es más divertido descubrir contenido, chatear en tiempo real, recomendar y compartirlo en otras plataformas relevantes para los usuarios como *Facebook* y *Twitter*.
- Descubrir el contenido de TV: Guía de programa social y recomendaciones.
- Social: check-in con *Facebook* y *Twitter* para compartir con los usuarios y chatear en tiempo real con usuarios de *Vuqio*.

1.2 Objetivos del proyecto

El objetivo de esta aplicación es hacer que el usuario tenga por un lado todas las funcionalidades de las aplicaciones que ya está usando para compartir contenidos cuando ve la televisión. Y por otro servicios de parrilla y de la audiencia en tiempo real. Todo esto contribuirá a mejorar la experiencia del usuario.
Los objetivos concretos son:

- Visualizado de la parrilla de programación actual y futura de la televisión a nivel nacional. Esta información vendrá dada por un proveedor externo.

- Servicio de mensajería instantánea para mantener conversaciones con otros usuarios registrados en *Vuqio* que vean o hayan comentado en *Twitter* cualquier programa emitido.

- Integración de redes sociales como *Twitter y Facebook* en la aplicación *Vuqio.* Respecto a la red social *Twitter,* el usuario será capaz de realizar acciones como *twittear, retwittear, responder, hacer favorito, ver el perfil de otro usuario de Twitter y seguir a otro usuario.* También se dará la ventaja de mostrar todos los

tweets relacionados de un programa sin tener que buscar por *hastags,* esto mejora la experiencia del usuario integrando todo el contenido en un único foco.

- Incorporación de Facebook para compartir contenido proporcionado por *Vuqio* como por ejemplo publicar en su tablón que programa y que opinión tiene respecto a este.

- Implementación de un gestor de alarmas para avisar de los programas y películas favoritos del usuario.

- Retransmisión de los programas emitidos en directo mostrando conceptos, lugares, personajes públicos, marcas y productos con los que el usuario podrá descubrir, a través de un navegador propio de la aplicación, enlaces con toda la información extraída del programa emitido o productos relacionados.

2. ESTUDIO DEL ESTADO DEL ARTE

2.1 Aplicaciones Relacionadas

La idea de *Vuqio* es anterior a las citadas a continuación pero han servido para crear una experiencia similar e incluso mejorarla. Se han elegido estas dos aplicaciones por varios motivos. Tockit es nuestra competencia directa debido a que es una aplicación a nivel nacional. Y en relación a Zeebox, que actualmente ha pasado a llamarse Beamly, es el modelo a seguir, debido a su gran éxito en EEUU, Reino Unido y Australia.

TOCKIT

La aplicación *Tockit* es de la misma temática pero incorporando *gamification* y haciendo *checkin* de los programas a través de la cámara del móvil. Además incorpora la posibilidad de ordenar la parrilla de televisión por audiencia en Twitter o Tockit, por canal y por chats abiertos. También permite hacer búsqueda de programas mostrándote la fecha de emisión de estos.

En comparación con Vuqio, Tockit gestiona los chat de forma diferente. En Tockit puedes crear chats públicos, en el cual puede entrar cualquier usuario y chats privados invitando a amigos tuyos de Tockit.

Vuqio permite al usuario saber de lo que se está hablando en un programa gracias al servicio de mostrar las palabras clave relacionadas con el contexto del programa (lugares, marcas, personajes, conceptos, acontecimientos, etc).

Figura 2.1 Icono Tockit

Figura 2.2 Aplicación Tockit.

Actualmente es la aplicación oficial del programa *Viajando con Chester* presentado por *Risto Mejide,* en el cual puedes participar en dicho programa haciendo una puja por un sofá.

ZEEBOX (BEAMLY)

La aplicación Zeebox ha sido el mayor referente para crear la aplicación Vuqio. En ella al igual que en Vuqio socializa la visión de la televisión ampliándola con la incorporación de las redes sociales. Provee al usuario de enlaces donde comprar aquello que está viendo en la televisión y compartir sus gustos televisivos con amigos.
La compañía fue fundada en Reino Unido en noviembre de 2011. En el 2012 el 70% de los televidentes de Gran Bretaña interactuaban con su dispositivo mientras veían la televisión por lo que Zeebox se convirtió en un referente.

Figura 2.3 Icono Zeebox.

Figura 2.4 Aplicación Zeebox.

En Abril de 2014 Zeebox pasa a ser Beamly, un servicio de red social para la televisión orientada a mujeres menores de 35 años. En ella se puede seguir a tus artistas y shows televisivos favoritos. Beamly no deja de ser una red social donde podremos encontrar a

otros usuarios con los que hablar sobre nuestras series favoritas, para ello la aplicación cuenta con diversas salas de chat o *TV-Rooms* para cada serie.

Conforme el usuario vaya indicando su gustos televisivos, la aplicación irá ofreciendo una guía de televisión personalizada y poder acceder a todo tipo de contenido e información.

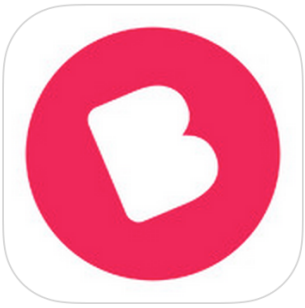


Figura 2.5 Icono Beamly.

Beamly se encuentra disponible en EEUU, Reino Unido y Australia.

Figura 2.6 Aplicación Beamly.

3. DEFINICIÓN DEL PROYECTO

3.1 Requerimientos funcionales

Los requisitos funcionales son las acciones fundamentales a las que dar respuesta el software.

- **RF1:** Guía de televisión.

  La aplicación debe permitir visualizar la parrilla de la programación televisiva.

  Este requisito se divide en:

  - **RF1.1:** El usuario cuando ejecute la aplicación obtendrá inmediatamente la guía de televisión.

  - **RF1.2:** Consultar la guía de televisión.

    Debe ofrece la posibilidad de navegar a través de la programación mediante dos formas:
    - Desplazamiento lateral de los programas de los distintos canales.
    - Especificando día y hora (Los dos días próximos al actual).

  - **RF1.3:** Información básica de los programas.

    En la ventana de la parrilla se mostrará por cada canal la imagen de este para que el usuario sepa que canal es. Además se ofrecerá la imagen, el titulo, hora inicial y final y progreso del programa actual.

  - **RF1.4:** Mostrar amigos.

    El usuario podrá ver en la guía de televisión los programas que están viendo sus amigos.

  - **RF1.5:** Servicio "En directo".

    Ciertos canales ofrecerán al usuario poder obtener información sobre que se está hablando en sus programas.

  - **RF1.6:** Popularidad de los programas.

    El usuario podrá ver el porcentaje de popularidad de los programas actuales en Twitter, es decir, los programas más comentados en Twitter.

- **RF1.7:** Programar alarma.

  El usuario podrá notificar que le avisen de programas futuros todos esto será detallado en el **RF9**.

- **RF2:** Información detallada del programa.

  Una vez que el usuario seleccione un programa de la guía de televisión, accederá a toda su información.

  Esto se divide en los siguientes requisitos:

  - **RF2.1:** Información básica del programa.

    Ya nombrada en los requisitos de la guía de televisión **RF1.3**.

  - **RF2.2:** Contenido en Twitter.

    Tweet relacionados con el programa y publicados por otros usuarios en Twitter o en Vuqio.

  - **RF2.3:** Información detallada del programa.

    - Usuarios de Twitter comentando el programa.
    - Número de Tweets relacionados con el programa.
    - Carrusel de Tweets relacionados con el programa.
    - Sinopsis del programa.
    - Interpretes del programa.
    - Carrusel de palabras claves obtenidas del programa.

- **RF3:** Registrarse mediante las redes sociales Facebook y Twitter.

  El usuario podrá sincronizar sus cuentas de Facebook y Twitter para poder utilizar distintos servicios que ofrece Vuqio como por ejemplo poder chatear con otros usuarios, compartir lo que esta viendo o crear sus avisos de comienzo de una emisión de un programa.

- **RF4:** Perfil del usuario.

  El usuario podrá visitar su propio perfil donde podrá:

  - **RF4.1:** Modificación de la biografía.

    Podrá incorporar un pequeña biografía para que pueda ser consultada por usuarios de Vuqio.

  - **RF4.2:** Gestión de las cuentas del usuario de Twitter y Facebook.

El usuario podrá loguearse o desloguearse de sus cuenta de Facebook o Twitter.

- **RF4.3:** Último programa visitado.

  El usuario podrá consultar su último programa visitado

- **RF5:** Twitter.

  Se permitirá conexión directa con Twitter y el usuario podrá a través de Vuqio:

  - Twittear sobre lo que esta viendo.
  - Retwittear a usuarios de Twitter que hayan comentado un programa.
  - Seguir/dejar de seguir a usuarios de Twitter.
  - Consultar perfiles de usuarios de Twitter.
  - Contestar a usuarios de Twitter.

- **RF6:** Facebook.

  El usuario podrá compartir lo que está viendo a través de Vuqio en Facebook. En la publicación aparecerá la imagen, el titulo y pequeña descripción del programa junto con el texto editado por el usuario.

- **RF7:** Chat.

  El usuario tendrá disponible dos tipos de chat.

  - **RF7.1:** Chat específico.

    Se mostrarán usuarios que estén viendo el mismo programa que el usuario (usuarios de Vuqio) y usuarios que no está registrados en Vuqio pero han comentado dicho programa en Twitter.

  - **RF7.2:** Chat global.

    Se mostrarán todos los usuarios de Vuqio primero y después todos los usuarios que han comentado en Twitter algún programa alguna vez.

    Si el usuario quisiera hablar con un usuario que no está registrado en Vuqio se le enviará una invitación a través de Twitter. Esto provocará la viralización de la aplicación.

  - **RF7.3:** Mostrar conversaciones abiertas.

    El usuario tendrá un listado de conversaciones que haya mantenido.

Icono que muestra el numero de usuarios que le han escrito y no ha visto.

- **RF7.4:** Borrar conversaciones.

  El usuario podrá borrar conversaciones.

- **RF7.5:** Bloquear/Desbloquear a usuarios.

  El usuario podrá bloquear usuarios de Vuqio y será borrada la conversación del listado de las conversaciones.

  Para desbloquear a un usuario bloqueado el usuario tendrá que iniciar la conversación.

- **RF8:** Etiquetas informativas.

  El usuario estará siempre informado sin estar viendo la televisión gracias a las etiquetas informativas.

  Para ello se divide en los siguientes requisitos:

  - **RF8.1:** Mostrar entidades.

    El usuario podrá seguir el programa sin necesidad de verlo gracias al servicio que proporciona Vuqio, donde se presenta al usuario un listado de conceptos extraídos del contexto del programa.

  - **RF8.2:** Mostrar los enlaces de las entidades.

    Si el usuario quiere obtener información de unos de los conceptos del programa se le proveerá de enlaces relacionados con el término (Google, Ebay, Wikipedia, Marca, Amazon, Spotify).

  - **RF8.3:** Navegador nativo.

    El usuario podrá consultar dichos enlaces sin necesidad de abandonar la aplicación.

- **RF9:** Alarmas.

  El usuario nunca podrá perderse ningún programa gracias al servicio de aviso de alarmas que le permitirá:

  - **RF9.1:** Crear alarma.

    El usuario podrá crear:

        - Alarma de un sólo programa que se emite en una fecha concreta.
        - Conjunto de alarmas de un programa que se puede emitir en distintos canales en distintas fechas.

    - **RF9.2:** Borrar alarma.
      El usuario podrá borrar:

        - Una alarma concreta de una emisión de un programa.
        - Todas las emisiones de un programa.

    - **RF9.3:** Cambiar hora de aviso.

      El usuario podrá cambiar el tiempo de inicio de aviso de la alarma, por defecto la alarma será mostrada media hora antes de que comience el programa.

    - **RF9.4:** Mostrar alarmas.

      Se mostrará todas las alarmas creadas por el usuario y ordenadas por la fecha de emisión.

3.2 Requerimientos no funcionales

Los requisitos no funcionales son los requisitos de interfaz, usabilidad, operacionales, de documentación, de rendimiento, de fiabilidad, seguridad, etc.

Los requisitos no funcionales son:

- **RNF1:** La aplicación está orientada a dispositivos táctiles con el sistema operativo Android a partir de la versión 2.3 (Gingerbread).
- **RNF2:** La aplicación funcionará solo si está conectada a internet.
- **RNF3:** La seguridad está garantizada gracias al cifrado de los datos del usuario y en las peticiones al *backend*.
- **RNF4:** Existirá mantenimiento de la aplicación tras las nuevas publicaciones de versiones del sistema operativo Android.
- **RNF5:** Diseño adaptable a los distintos tamaños de pantalla y densidad.
- **RNF6:** Control de errores para garantizar la estabilidad de la aplicación.
- **RNF7:** Al abrir por primer vez la aplicación se proporciona un tutorial con la información básica de Vuqio.
- **RNF8:** Caché de imágenes y peticiones para mejorar el tiempo de respuesta.
- **RNF9:** La aplicación no necesita estar en segundo plano gracias al envío de notificaciones [2][3].

4. Diseño de la solución

Para la aplicación Vuqio se ha estudiado y realizado un diseño que recoge todos los requerimientos descritos en el apartado 3.

4.1 Diseño de la interfaz

Para el diseño de la interfaz se ha procurado crear pantallas que respondan a todos los requisitos y que hagan que la aplicación sea intuitiva.

El diseño mostrado en la Figura 4.1 engloba toda la navegación permitida para un usuario que se descargue la aplicación y no se registre en Vuqio.

Por otro lado la Figura 4.2 muestra el diseño de la aplicación para aquellos usuarios registrados en Vuqio.

Los bloques comunes a las dos Figuras citadas anteriormente son:

- Menú Lateral (Bloque 1): Contiene a los bloques 2 y 3 y será mostrado en todas las pantallas de dichos bloques.
- Guía de televisión (Bloque 2): En la sección 4.1.1 se describe en detalle este bloque.
- Información detallada del programa (Bloque 3): En la sección 4.1.2 se describe en detalle este bloque.

Debido a la gran complejidad de la navegación de la interfaz a continuación se va a detallar los aspectos más importantes considerando que el usuario ya se ha registrado. Se dividirá en 4 bloques Guía de Televisión y Chat, Información detalla del programa, Gestión de Alarma y Notificaciones.

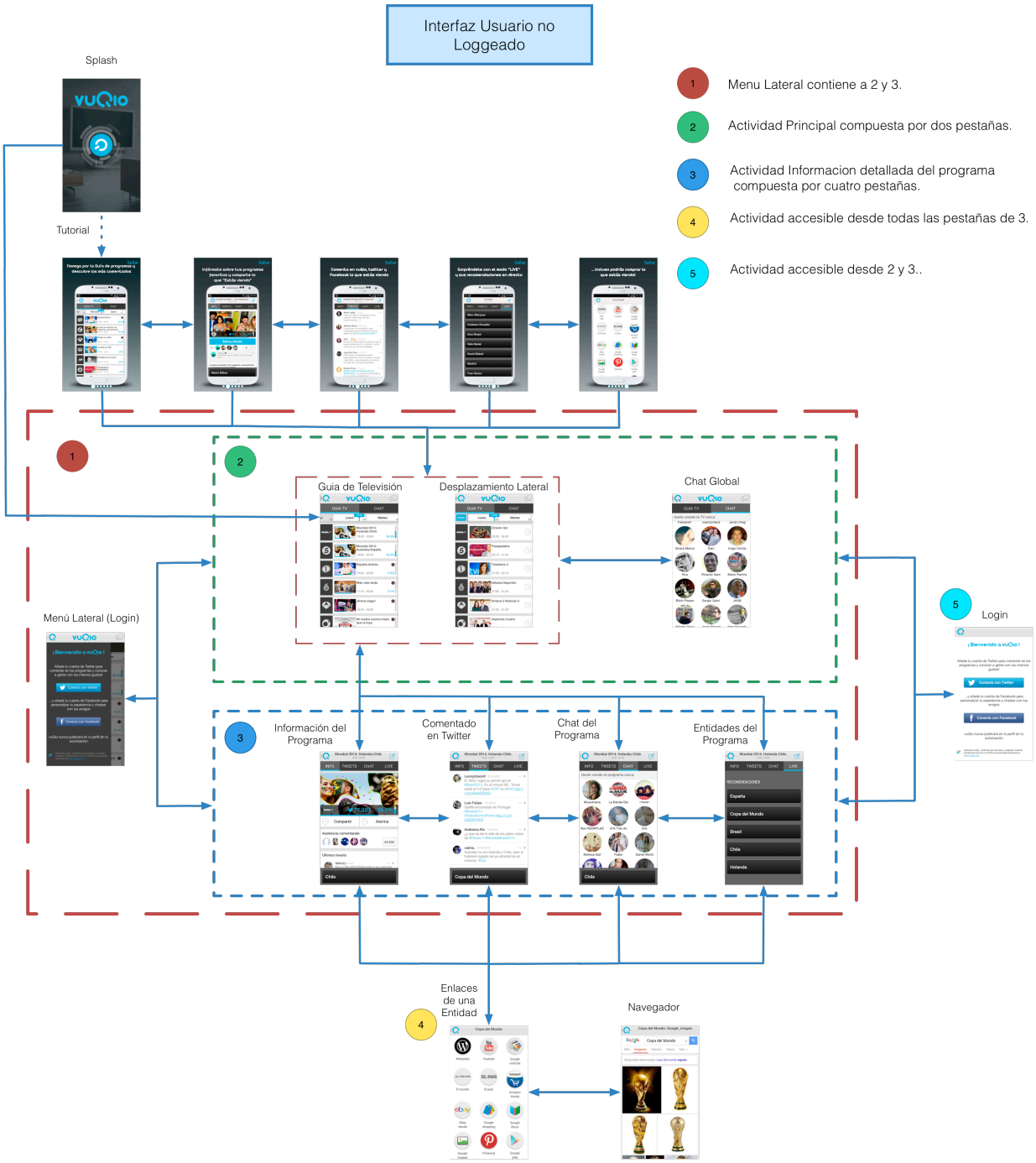
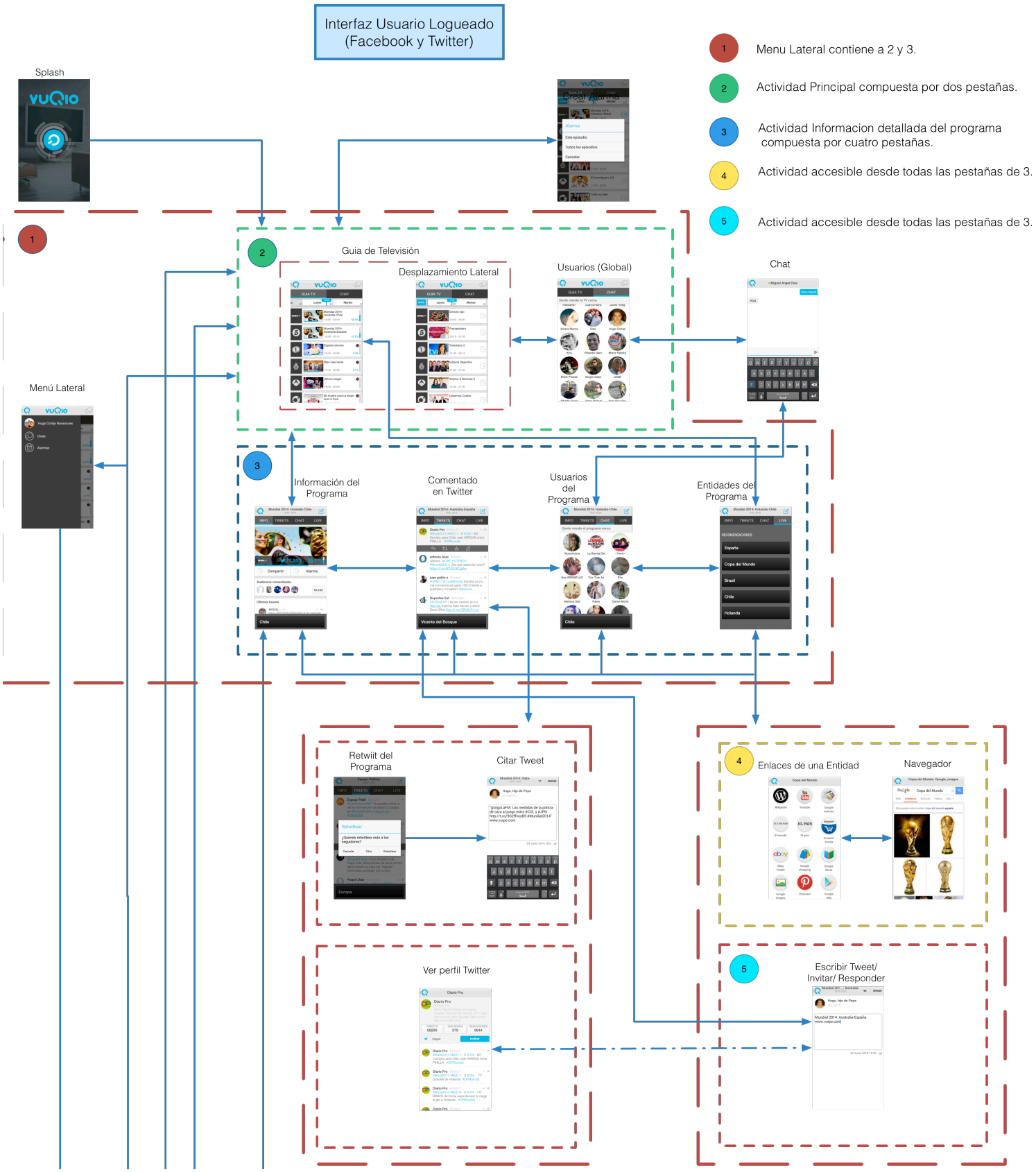


Figura 4.1 Diseño de la Interfaz del Usuario no logueado.


Interfaz Usuario Logueado
(Facebook y Twitter)
1 Menu Lateral contiene a 2 y 3.
2 Actividad Principal compuesta por dos pestañas.
3 Actividad Informacion detallada del programa compuesta por cuatro pestañas.
4 Actividad accesible desde todas las pestañas de 3.
5 Actividad accesible desde todas las pestañas de 3.
Splash
Guia de Televisión
Desplazamiento Lateral
Usuarios (Global)
Chat
Menú Lateral
Información del Programa
Comentado en Twitter
Usuarios del Programa
Entidades de Programa
Retwiit del Programa
Citar Tweet
Enlaces de una Entidad
Navegador
Ver perfil Twitter
Escribir Tweet/ Invitar/ Responder

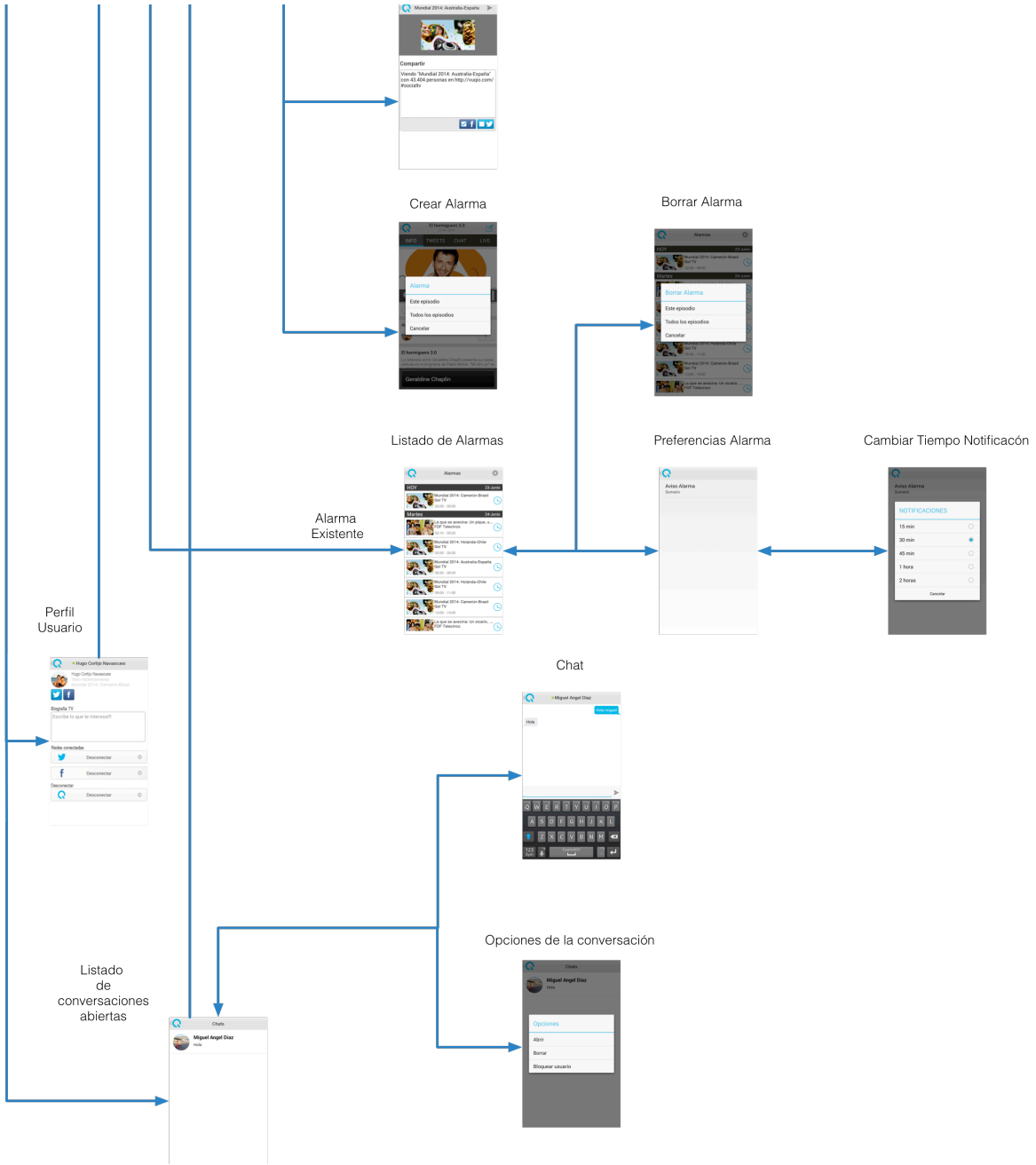


Figura 4.2 Diseño de la Interfaz del Usuario logueado.

4.1.1 Interfaz principal: Guía de Televisión y Chat Global.

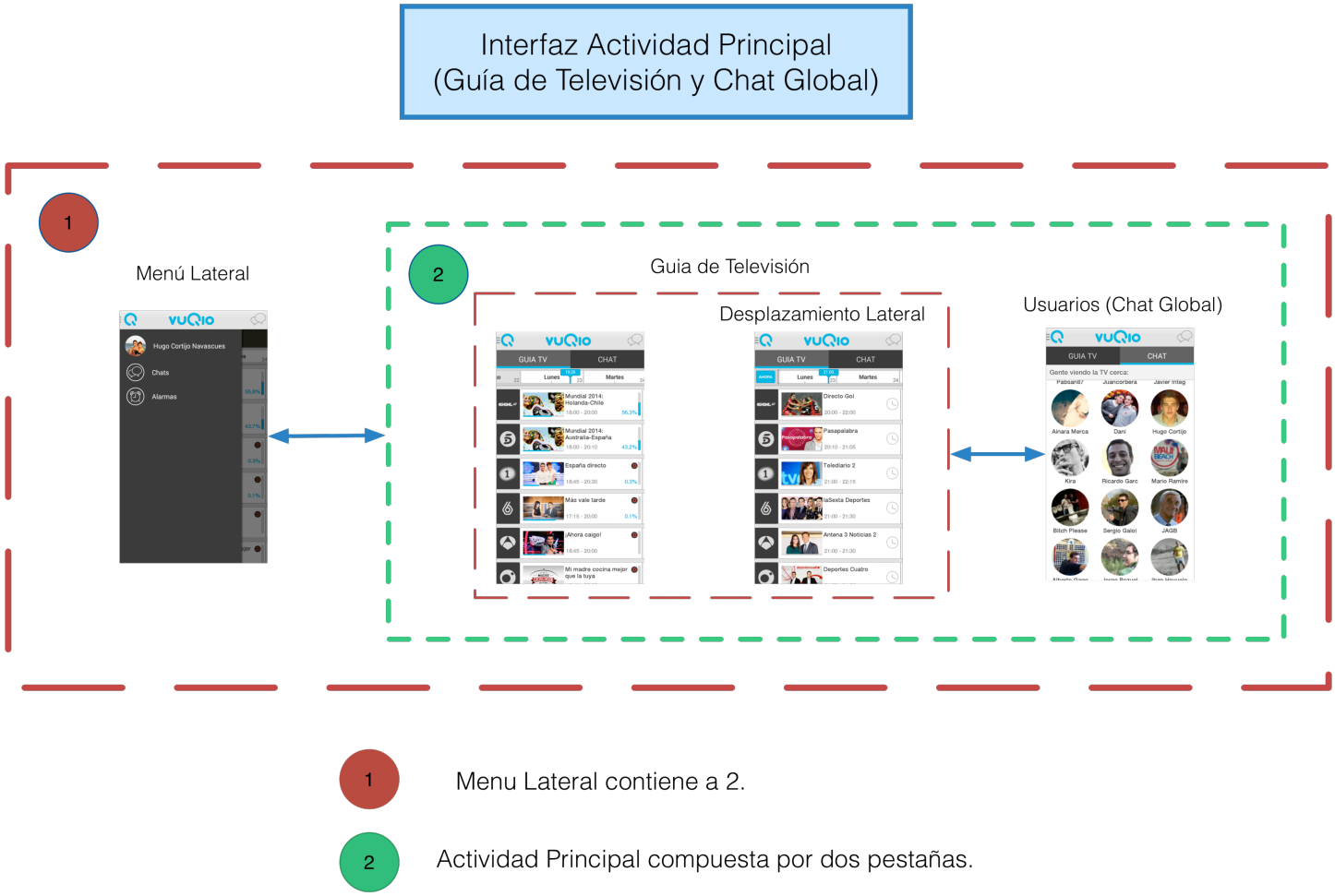


Figura 4.3 Actividad Principal de Vuqio.

La interfaz principal está compuesta por dos pestañas:

- **Guía de Televisión**: Por defecto siempre se mostrará dicha interfaz. En ella se cumplen los **RF1** especificados en el apartado 3.1. Si el usuario desplaza lateralmente de derecha a izquierda o viceversa alguno de los ítems de la lista de programación televisiva se modificará la guía mostrando programas futuros o pasados respectivamente. Si el programa que se muestra en la parrilla ya ha terminado o no ha comenzado no se mostrará popularidad en Twitter. Y si el programa no ha comenzado se añade a la vista el icono de alarmas para poder gestionarlas. Si el usuario pulsa el "icono alarma" y este ya estaba activo (alarma creada previamente) le llevaría a la lista personalizada de alarmas del usuario, si no fuera así se le muestra al usuario si desea crear una alarma (**RF9.1**).
  También a través de la guía, el usuario puede ir a la interfaz de información detalla del programa (apartado 4.1.2) al pulsar en alguno de los ítems de la lista de la programación o en los icono de los canales.

Otra forma de consultar la guía de televisión es deslizando o pulsando una posición de la barra horizontal de días mostrada a continuación:

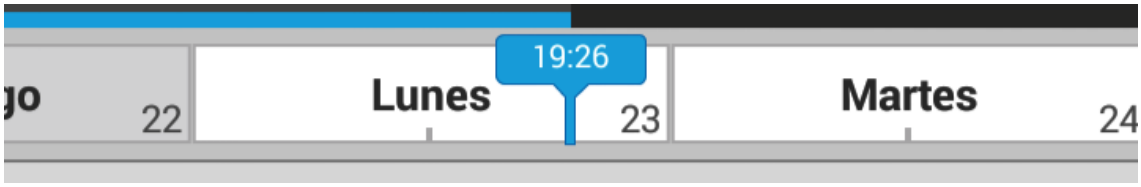


Figura 4.4 Indicador actual de la Guía de Televisión

Mediante el elemento mostrado en el Figura 4.4 el usuario podrá consultar la guía de televisión especificando el día y hora. Si el usuario consulta otra hora que no sea la actual la vista se actualizará como se puede ver en la Figura 4.5, añadiéndose un botón a la izquierda que le permita al usuario volver rápidamente a la fecha actual.

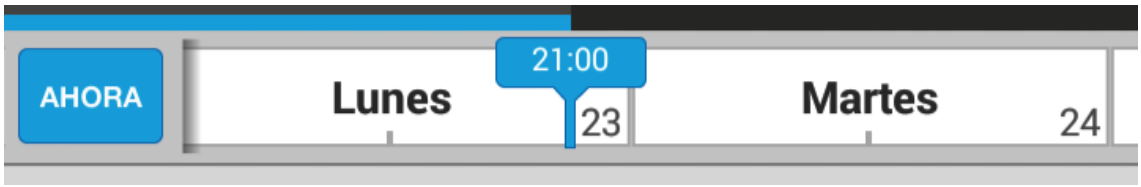


Figura 4.5 Acceso automático a la fecha actual de la Guía de Televisión

- **Chat Global:** En esta pestaña se da respuesta al **RF7.2** del apartado 3.1. Si el usuario quiere chatear con otro usuario y no está logueado se le mostrará la interfaz de registro (Figura 4.6) y una vez que el usuario se haya logueado con Facebook o Twitter se abrirá la interfaz de mensajería (Figura 4.7). Si el usuario estuviera ya logueado iría a la interfaz mostrada en la Figura 4.7.

Figura 4.6 Interfaz de registro en Vuqio.

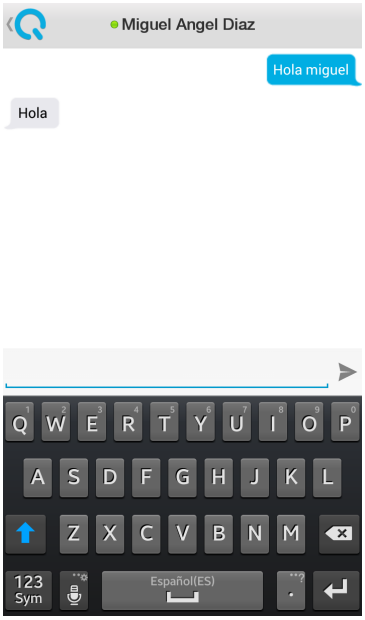


Figura 4.7 Interfaz de mensajería en Vuqio.

Si el usuario deseara hablar con un usuario que ha comentado algo en Twitter pero que no está registrado en Vuqio, igualmente se abrirá la interfaz de mensajería pero los mensajes se almacenarán en el *backend* y se le notificará vía Twitter a ese usuario que alguien de Vuqio quiere hablar con el e invitándole a que se baje la aplicación. Una vez que el nuevo usuario se descargue la aplicación y se registre le llegarán todos los mensajes enviados por otros usuarios de Vuqio. Esta funcionalidad tiene el objetivo añadido de difusión de la aplicación entre potenciales usuarios.

Aparte de las dos pestañas la interfaz principal tiene un menú lateral que permite al usuario registrarse si no está logueado (Figura 4.8) y si lo está acceder a su perfil, alarmas y conversaciones abiertas (Figura 4.9). Para mostrar el menú lateral solo es necesario deslizar el dedo sobre la pantalla desde el lado izquierdo hacia el derecho o pulsando el icono de la parte superior izquierda.

Figura 4.8 Interfaz menú lateral no logueado.

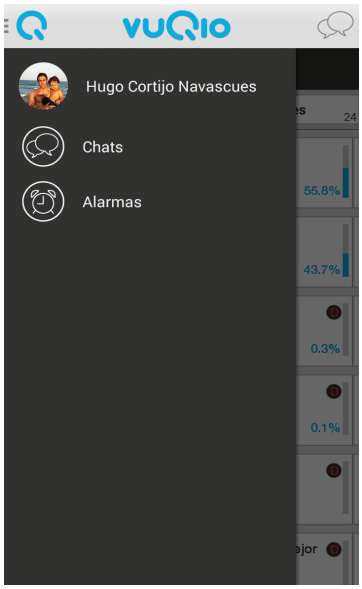


Figura 4.9 Interfaz menú lateral logueado.

El usuario, ya logueado, que recibe un mensaje de algún otro usuario de Vuqio se le notificará en la interfaz principal como se puede ver en la Figura 4.10 y al pulsar el icono se mostrará las conversaciones abiertas.

Figura 4.10 Notificación de conversaciones no leídas.

4.1.2 Interfaz información detallada del programa

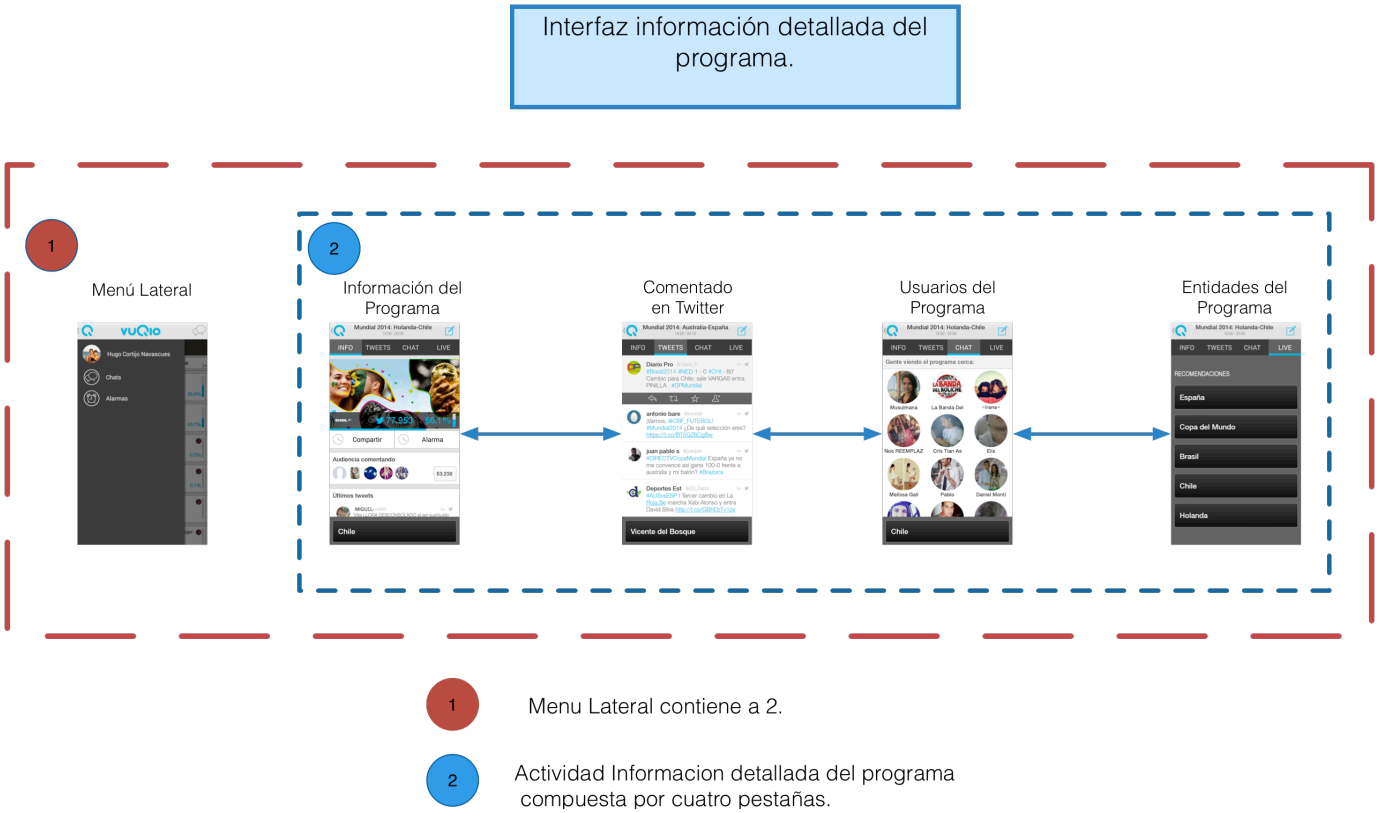


Figura 4.11 Interfaz de la información detallada del programa

En la Figura 4.11 se muestra la interfaz de la información detallada del programa. A esta interfaz se accede seleccionando uno de los programas mostrados en la parrilla de la televisión. Este panel de la aplicación contiene cuatro pestañas: Información, tweets, chat y entidades del programa. Al igual que en la interfaz principal, se puede mostrar el menú lateral pero esta vez sólo será visualizado al deslizar el dedo desde la parte izquierda de la pantalla hacia la derecha. Además del menú lateral aparece una vista común a las cuatro sub-interfaces mostrada en la Figura 4.12.

Figura 4.12 Elemento común a las cuatro pestañas.

A continuación se detallan las cuatro sub-interfaces de esta interfaz:

- **Información del programa:** En esta sub-interfaz se satisfacen los requisitos **RF2.1** y **RF2.3.** Respecto a la gestión de alarmas si el programa está en emisión se podrán crear alarmas para emisiones futuras siempre y cuando la alarma no exista previamente. Si la alarma ya está registrada se le mostrará al usuario su listado de alarmas (**RF9.1).** Si el programa tiene tweets relacionados se mostrará un carrusel que cada 5 segundos irá cambiando de tweet y además el usuario podrá recorrerlo deslizando el dedo sobre él. Si el usuario decide pulsar el carrusel se le dirigirá a las siguiente pestaña "Comentado en Twitter" (Figura 4.13). Para concluir el usuario podrá compartir en Facebook y Twitter lo que está viendo si pulsa el botón "Compartir" (Figura 4.13).

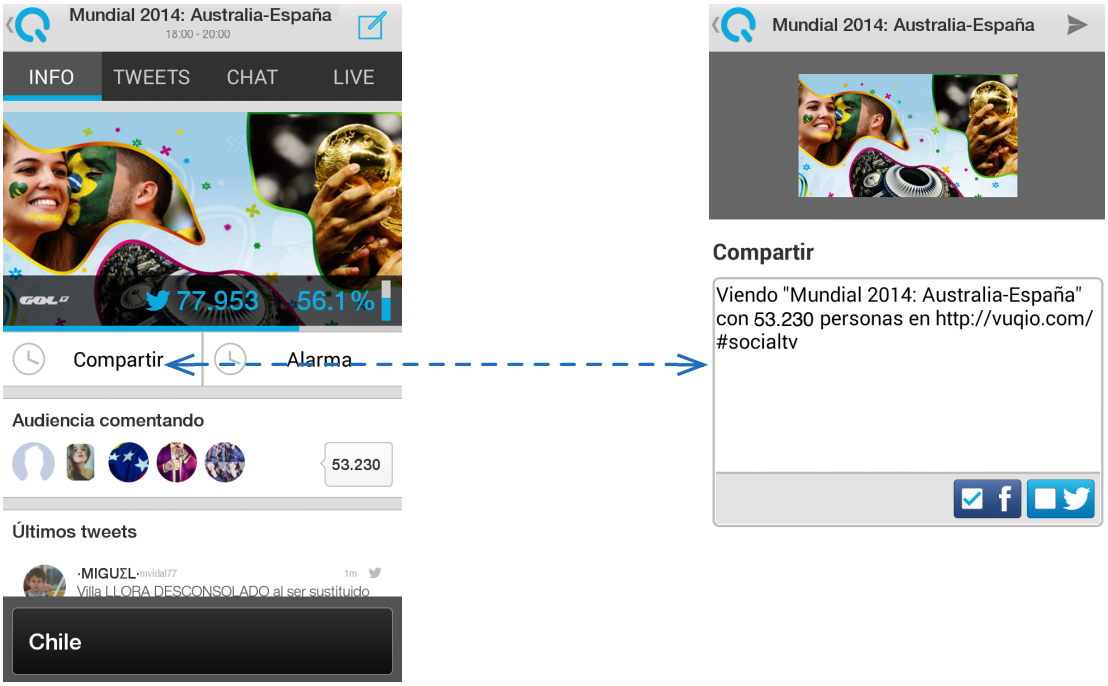


Figura 4.13 Compartir en Facebook y Twitter.

- **Comentado en Twitter:** En esta pestaña se cumplen los requisitos **RF2.2** y **RF5.1.** El usuario podrá ver y actualizar la lista de comentarios de otros usuarios que han compartido en Twitter comentarios sobre el programa en cuestión (lista máxima de 20 tweets por defecto). La actualización de la lista se produce al deslizar el dedo desde la parte superior de la lista hacia abajo. Si el usuario quiere leer tweets anteriores deberá ir al final de la lista para que se carguen los tweets previos. Si el usuario desea realizar algunas de las acciones más características de Twitter como "Citar", "Retwitear/Borrar el Retweet", "Responder", "Seguir/Dejar de Seguir" y "Hacer/Deshacer favorito" deberá seleccionar algún ítem de la lista de tweets y se mostraran todas las acciones citadas anteriormente (Figura 4.14).

Figura 4.14 Opciones de Twitter disponibles en Vuqio.

El usuario puede consultar el perfil de un usuario de Twitter pulsando la imagen de este (Opción 5 Figura 4.14) y se mostrará toda información obtenida de Twitter de dicho usuario (Figura 4.15).

Figura 4.15 Perfil de usuario de Twitter en Vuqio.

- **Chat del programa:** Los requerimiento que cumple esta sub-interfaz corresponden al apartado **RF7.1.** Gracias a esta pestaña el usuario podrá hablar con aquellas personas que hayan entrado o comentado un programa concreto. La lógica es la misma que el chat global de la interfaz principal de la aplicación.

- **Descubre el programa:** En esta pestaña se muestra al usuario las entidades más importantes que se han nombrado en el programa gracias al procesado de audio y de subtítulos de este (Figura 4.16). Aparte de entidades dinámicas habrá entidades estáticas no relacionadas con el audio o subtítulos del programa, todas ellas estarán expuestas en una lista que se actualiza cada 15 segundos o cuando el usuario quiera simplemente deslizando el dedo desde la parte superior de la lista hacia abajo (Figura 4.17). Esta sub-interfaz está descrita en el apartado 3.1.

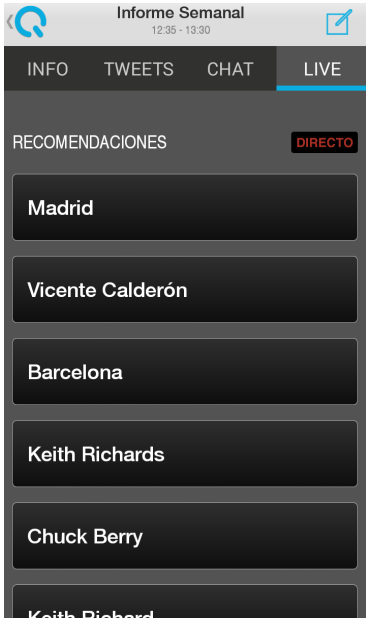


Figura 4.16 Interfaz lista de entidades.

Figura 4.17 Actualizar lista de entidades.

Como se puede ver en la Figura 4.11 las tres primeras pestañas tienen un elemento común que es el carrusel de las entidades que se muestran en la pestaña "LIVE" con un formato diferente. El usuario puede pasar las entidades deslizando el dedo sobre el carrusel en dirección horizontal. Si el usuario selecciona alguna entidad del carrusel se mostrará todos los enlaces relacionados (**RF8.2**) con dicha entidad como se puede ver en la Figura 4.17.

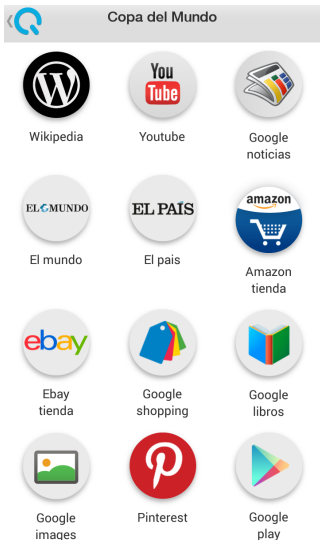


Figura 4.18 Interfaz lista de enlaces de una entidad.

Figura 4.19 Navegador de Vuqio.

Como se muestra en la Figura 4.18 al pulsar en este caso el icono “Google Imágenes” la aplicación Vuqio abre un navegador propio donde visualizar el enlace (Figura 4.19).

4.1.3 Interfaz menú lateral

Gracias al menú lateral el usuario podrá acceder a su perfil, a su lista de alarmas y a sus conversaciones abiertas. Como se ha dicho anteriormente para acceder a este menú hay que deslizar el dedo desde la parte izquierda de la pantalla hacia a la derecha.
A continuación se muestra que accesos tiene el usuario a través del menú:

4.1.3.1 Perfil del usuario

El usuario podrá editar su biografía y sus cuentas de Twitter y Facebook **RF4.1.**

4.1.3.2 Gestión de alarmas

Esta interfaz (Figura 4.20) contendrá una lista de alarmas ordenadas por fecha de comienzo del programa previamente programadas por el usuario **RF9.4**. Habrá dos tipos de alarmas:

- **Alarma única:** Alarma que notifica de una sola emisión.
- **Alarmas de un programa:** Si el usuario le gusta una serie, programa o película podrá seguir todas las emisiones y será notificado en que canales se emite dicho programa, serie o película.

Si el usuario pulsa el icono de alarma de alguno de los ítems de la lista podrá borrar alarmas **RF9.2.**

Figura 4.20 Interfaz gestión de alarmas

Para poder editar el aviso de la emisión de un programa (por defecto media hora antes del comienzo del programa) sólo basta con pulsar el icono de la parte superior derecha. Y se mostrará un menú el cual contiene varias opciones (Figura 4.21).

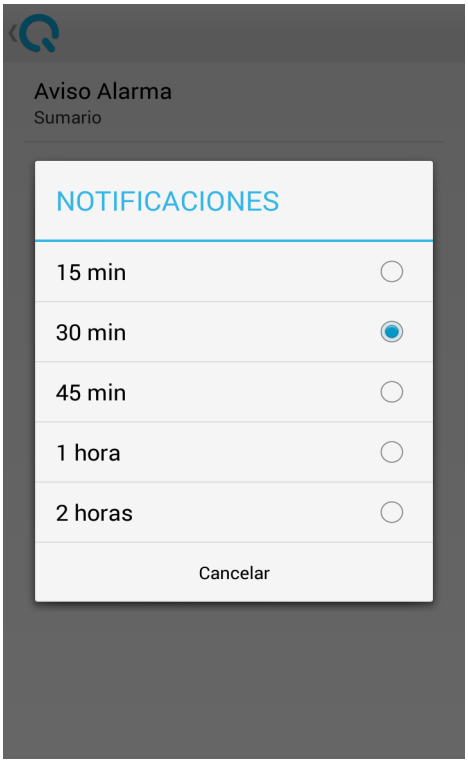


Figura 4.21 Opciones de tiempo de aviso previo.

El usuario podrá desde la lista de alarmas ir a la interfaz de información detalla del programa pulsando cualquiera de los ítems de la lista.

4.1.3.3 Gestión de conversaciones abiertas

En este bloque de la interfaz se mostrará una lista de conversaciones abiertas ordenadas primero por aquellas sin leer y después el resto (**RF7.3)**.

Si el usuario mantiene la pulsación en cualquiera de los ítems de la lista de conversaciones abiertas (Figura 4.22) aparecerá un menú (**RF7.4** y **RF7.5**) y si la pulsación no es prolongada se abrirá la interfaz de mensajería (Figura 4.7).

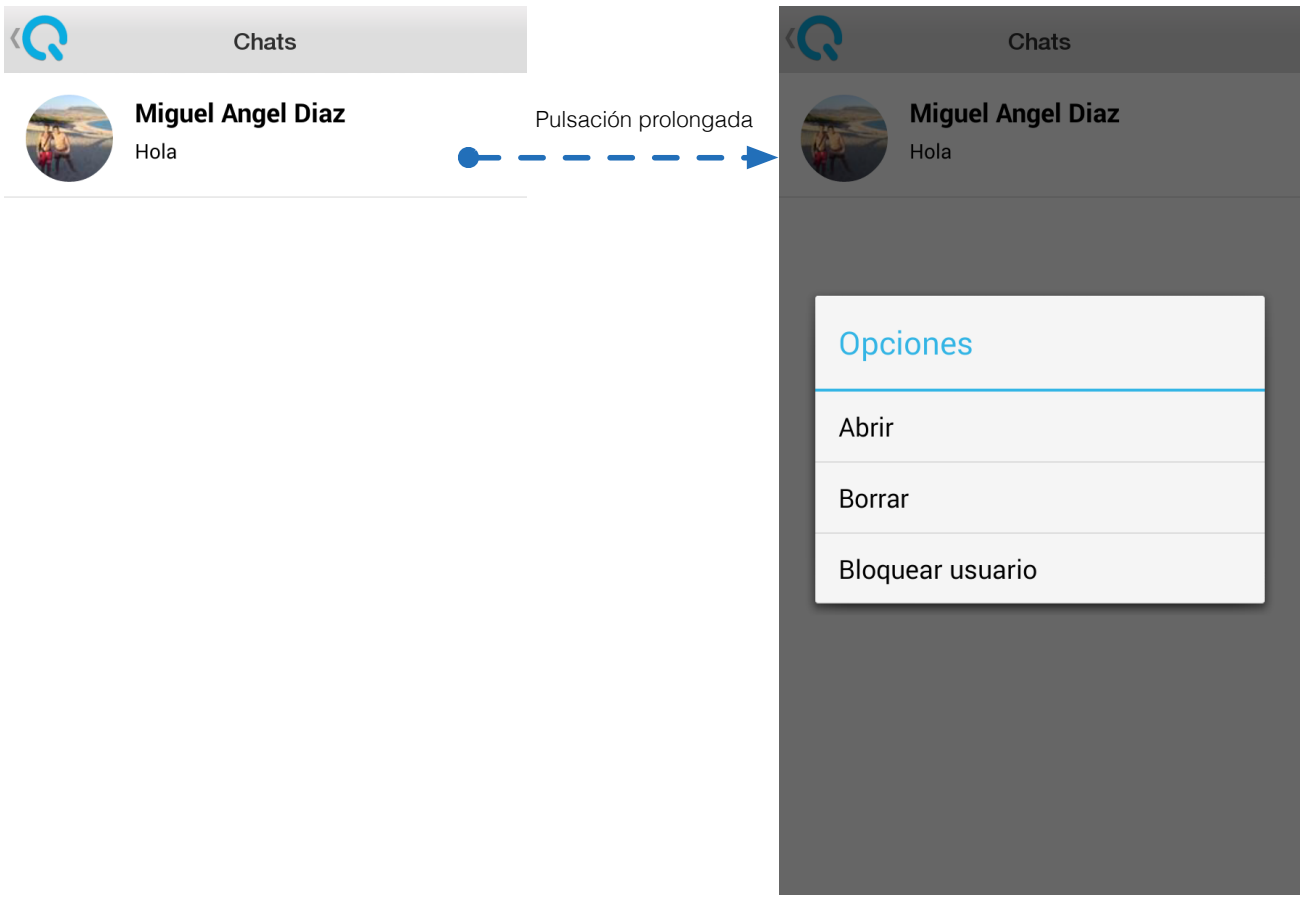


Figura 4.22 Interfaz gestión de conversaciones abiertas.

4.1.4 Interfaz notificaciones

En el caso de tener la aplicación cerrada tanto para las alarmas como mensajes nuevos el usuario tiene que ser notificado. Para ello se mostrarán notificaciones a través del sistema operativo del móvil Figura4.23.

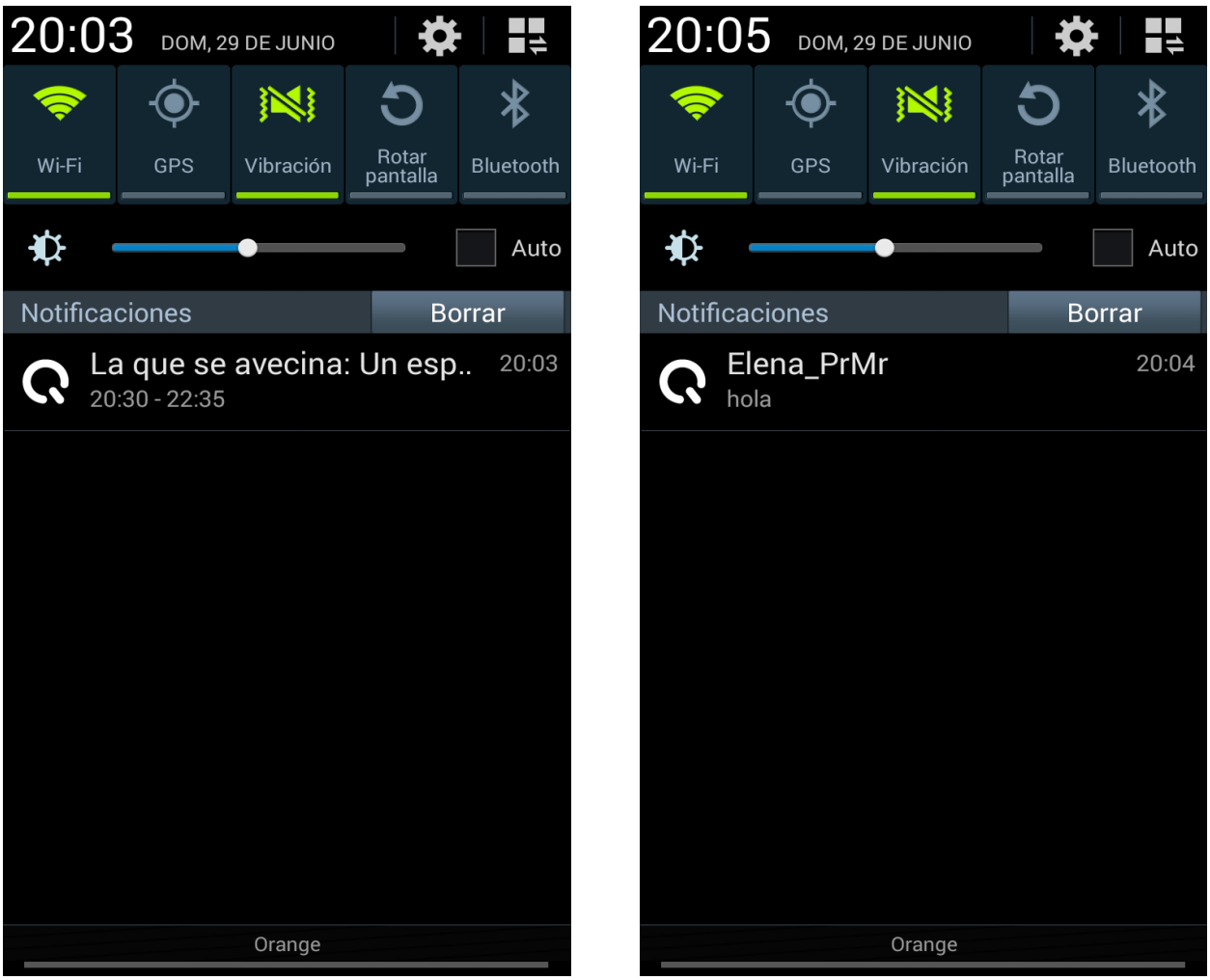


Figura 4.23 Interfaz de las notificaciones alarma y chat.

4.2 Diseño de la base de datos

A continuación se muestra el diseño de la base local del dispositivo para gestionar tanto las conversaciones abiertas como las alarmas creadas.

4.2.1 Chat

4.2.1.1 Diseño Preliminar

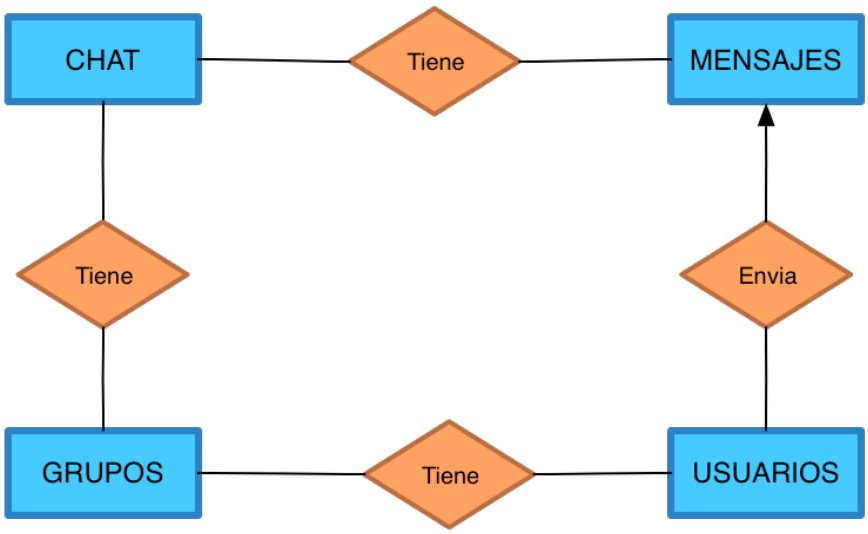


Figura 4.24 Entidad Relación Chat

Inicialmente se consideró la posibilidad de crear chat grupales y se dio la solución mostrada en la Figura 4.24. Finalmente como se muestra en el diagrama de la Figura 4.25 por motivos de planificación y prioridades en el *backend* se rediseñó la solución sólo permitiendo al usuario tener chats individuales.

4.2.1.2 Diseño Final

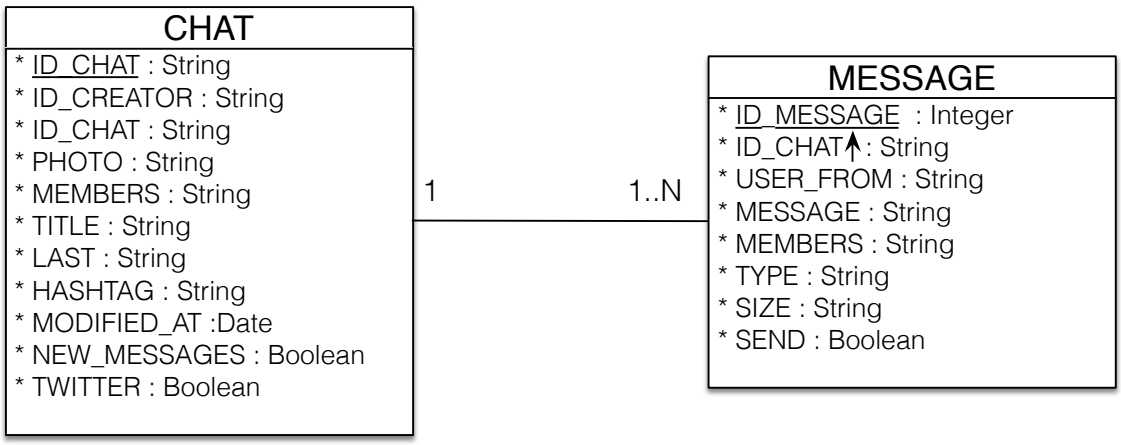


Figura 4.25 Tablas para la gestión del chat.

El diseño final de la base de datos para resolver la gestión del chat se muestra en la Figura 4.25. Consta de dos tablas, la tabla Chat guarda las conversaciones del usuario y esta se relaciona con la tabla Messages con una clave foránea ID_CHAT. Dicha tabla guardará los mensajes relacionados con un chat.

Uno de los campos más importantes es *twitter*. Sirve para identificar si se inicia un chat con algún usuario de Twitter no registrado en Vuqio ya que en el *backend* hay que notificarlo. Si en un futuro el usuario no registrado se descarga la aplicación y se loguea será notificado.

Otro campo importante es new_message para saber que chats no han sido leídos por parte del usuario.

4.2.2 Alarmas

4.2.2.1 Diseño Preliminar

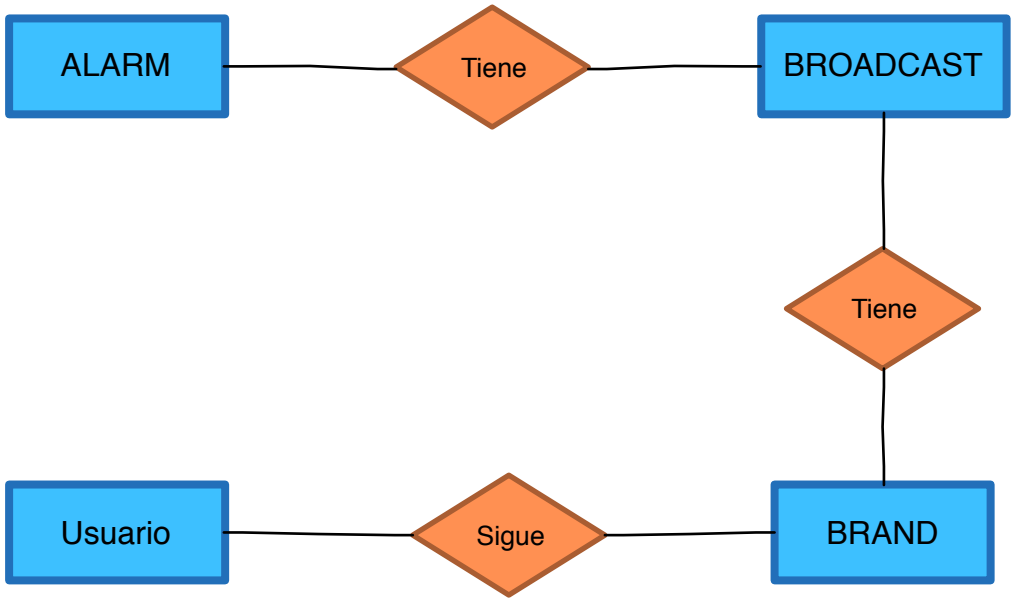


Figura 4.26 Entidad Relación Alarmas.

El diseño está basado en el modelo propuesto por la BBC [4] el cuál describe cómo las marcas, series, episodios y emisiones interactúan entre sí. Dicho modelo está desarrollado en el *backend* de forma rigurosa, pero en el desarrollo de la aplicación no se ha necesitado tanta complejidad.

En la Figura 4.27 se muestra el diseño final compuesto por una tabla llamada Follow Brand que guardará los IDs de las emisiones que seguirá el usuario. La tabla Program que guardará la información básica del programa y la tabla Alarm relacionada con la tabal Follow Brand y programa gracias a dos claves foráneas donde a continuación se citarán:

Las columnas "ID_FOLLOW_BRAND" de la tabla "Follow Brand" y "ID_BROADCAST" de la tabla "Alarm" permitirán la correcta eliminación en cadena de todas las emisiones que siga el usuario. Respecto a la eliminación de alarmas también tiene importancia el campo "DELETED" para actualizar la lista de alarmas de

usuario y no volver a mostrarlas a no ser que si se produce un error al realizar la petición de borrado de la alarma.

Debido a que el proveedor de la guía de televisión no es muy fiable en la programación televisiva futura, los campos "ID_BROADCAST", "START_DATE" y "END_DATE" de un programa puede cambiar, por lo tanto gracias a "ID_ALARM" generado por el *backend* y no modificado se pueden actualizar o eliminar alarmas. Dichas actualizaciones se realizarán en segundo plano de forma transparente al usuario.

4.2.2.2 Diseño Final

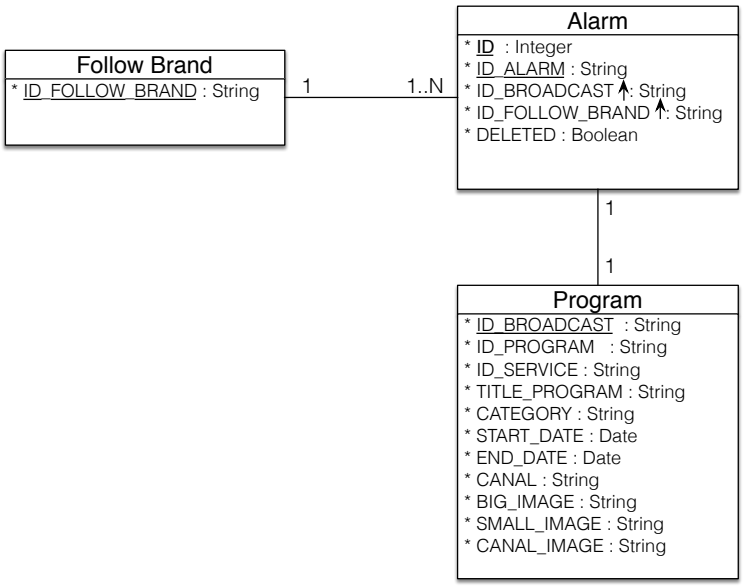


Figura 4.27 Tablas para la gestión de las alarmas.

4.3 Arquitectura del sistema

La arquitectura del sistema de Vuqio no forma parte de este trabajo de fin de grado aunque la conexión con ella sí. Es una arquitectura compleja de la que se detallan los dos bloques más importantes.

Uno de los bloques mas importantes es el servidor *API REST* escalable gracias a *NGINX* [5] ya que si aumenta la carga de peticiones realiza copias del servidor *API REST*.
El servidor *API REST* es el encargado de las peticiones realizadas por el usuario. Le ofrece una interfaz *API REST* en la que el usuario podrá, por ejemplo, hacer login, gestionar alarmas, etc. También se gestiona los *websockets* para el servicio de mensajería instantánea ofrecido a los usuarios de Vuqio. Los *websockets* se gestionan a través de los actores de *AKKA* [6]. Este servidor se conecta a los servicios del servidor estático.

El servidor de procesamiento es otro de los bloques más importantes de esta arquitectura. En él se cachean las peticiones más frecuentes de los usuarios a través de *Redis* [7]. Además accede a una base de datos *NoSQL* llamada *MongoDB* [8] que

permite indexar cualquier de los campos de la base de datos para mejorar el tiempo de consulta.

También este servidor recibe la información de la programación servida por un proveedor. Dicha información es enviada a las (02:00 am)dos de la madrugada por parte del proveedor a través del lenguaje de intercambio de datos *XML* [9]. Y una vez recibido el *XML* se lanza un *cron* que pide a *Ruby* [10] que haga el parseo e introducción en la base de datos de la información obtenida. Una vez completada la acción de procesamiento se generan los *JSON* [11] estáticos para la guía de televisión y se suben al servidor estático de Amazon S3 [12].

Otra parte muy importante en este servidor es el procesamiento de información en Twitter de los programas emitidos gracias a la API de Twitter Streaming [13]. La API de streaming de Twitter tiene muchas restricciones de las cuales dos han influido en el diseño de la arquitectura de Vuqio. La primera de las restricciones es que Twitter solo permite escuchar 400 *hashtag* y si se necesita modificar la lista donde se encuentran los *hashtag* hay que volver a reiniciar la conexión. Como Vuqio tiene aproximadamente 2000 *hashtag* diarios se define una ventana de cuatro horas para escuchar *hashtags* asociados a un programa antes y después de la emisión de este. Además se define un intervalo de media hora que se encarga de comprobar los *hashtags* que se están escuchando y se van a escuchar. Si hay *hashtags* diferentes se actualiza la lista y se reinicia el streaming. El problema que surge aquí es que cuando hay eventos con una gran carga en Twitter muchas reconexiones son fallidas. Si las reconexiones fallida se hicieran una detrás de otra nunca se reconectaría y por lo tanto habría una gran pérdida de tweets. Para solucionar esto se han creado dos hilos *AKKA*. Uno de ellos sigue el patrón anteriormente descrito pero los intervalos de conexión fallidos provocarán que las reconexiones se hagan en un tiempo logarítmico. Y el otro escucha los 400 hashtags más importantes, a criterio de Vuqio, durante las 24 horas.

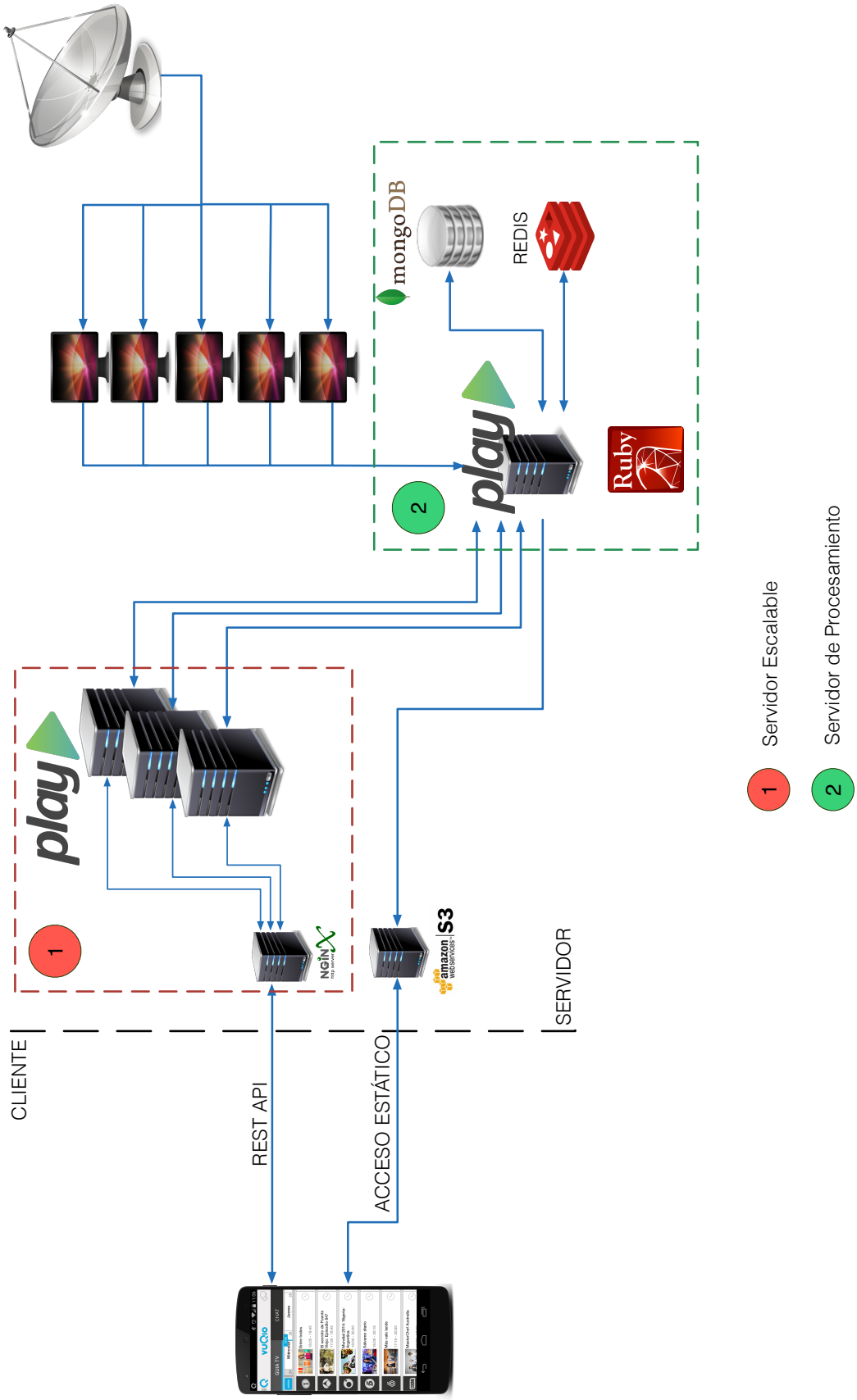


Figura 4.28 Arquitectura del sistema.

4.4 Tecnologías utilizadas

4.4.1 ANDROID SDK

Android SDK [14] (Software Development Kit) incluye un conjunto de herramientas de desarrollo y bibliotecas API para crear, probar y depurar aplicaciones para Android. Incluye una versión del IDE de Eclipse con una función de *ADT* (Android Development Tools) para agilizar el desarrollo de tu propia aplicación.

El ADT incluye:

- Eclipse + ADT plugin.
- Android SDK Tools.
- Android Platform-tools.
- La última plataforma Android.
- La imagen más reciente del sistema Android para el emulador

4.4.2 ECLIPSE

Conjunto de herramientas de programación de código abierto multiplataforma. En mi proyecto esta integrado con los plugins:

- JDK (Java Development Tools) para programar en Java.
- ADT (Android Development Tools) para desarrollar aplicaciones Android.

4.5 Seguridad

Debido al gran número de peticiones *HTTP* que involucran el id del usuario y por lo tanto toda su información habrá que desarrollar una capa de seguridad. Se utilizará una criptografía simétrica entre el dispositivo móvil del usuario con el *backend* para cifrar el id del usuario y del dispositivo asociado y no ser vulnerable a ataques.

Por lo tanto se ha decidido cifrar los datos mediante *AES128*+Salt ya que hoy en día computacionalmente es imposible mediante fuerza bruta descifrar la información encriptada. Además los tokens encriptados tendrán una fecha de caducidad imposibilitando un ataque *MitM*.

5. Solución

5.1 Elementos básicos

- **AndroidManifest.xml:** En Android, es un fichero de configuración de la aplicación que se vaya a desarrollar.
- **Action Bar:** El action bar de Android es la barra que aparece en la parte superior de la interfaz de la aplicación. Normalmente se muestra un icono, el titulo de la Actividad en la que se encuentra el usuario, botones de acción y un menú desplegable donde se muestra aquellos botones de acción que no tienen espacio.
- **Activity:** En Android, una Actividad es la encargada de crear la interfaz con la que el usuario va a interaccionar. La actividad tiene un ciclo de vida el cual hay que controlar.
- **Fragment:** En Android, un Fragment es una parte de la interfaz global de una pantalla. Esto permite modularizar mejor el código del programador y crear interfaces más complejas entre las pantallas grandes y pequeñas de los diversos dispositivos. El Fragment tiene que estar integrado en una Activity. Y al igual que la Activity tiene su propio ciclo de vida y es afectado por el ciclo de vida de la Activity.
- **FragmentActivity:** En Android, un FragmentActivity es una subclase de Activity que se introdujo en el paquete de compatibilidad *android-support.* En FragmentActivity se añadieron dos métodos para garantizar dicha compatibilidad con versiones anteriores de Android.
- **View Group:** En Android, un View Group es una vista que puede contener otras vistas denominadas hijos.
- **ListView:** En Android, un ListView es una vista que muestra los ítems de la lista verticalmente.
- **ExpandableListView:** En Android, un ExpandableListView es como el ListView pero al pulsar alguno de los ítems de la lista se expande mostrando a sus hijos.
- **ViewPager:** En Android, un ViewPager es una vista que permite al usuario desplazarse por las diferentes interfaces.
- **Adapter:** En Android, un Adapter actúa como un enlace entre la vista contenedora y los datos para esta vista. El Adapter proporciona acceso a los elementos de datos. También se encarga de hacer una vista por cada ítem del conjunto de datos.
- **Custom Adapter:** En Android, se puede customizar un Adapter cargando una vista diseñada al gusto del programador.

- **CustomView:** En Android, hay a veces que las vistas que proporcionan no cubren los requisitos que se le piden al programador. Además una custom view permite al programador crear vistas más robustas y reutilizables.

  Otro aspecto importante a la hora de utilizar custom views es la *retro compatibilidad* en un aplicación. Esto significa que la aplicación tendrá siempre el mismo aspecto para versiones anteriores.

- **HorizontalScrollView:** En Android, un HorizontalScrollView es un contenedor para una jerarquía de vistas que se pueden desplazar y puede ser más grande que la propia pantalla física. Es un *FrameLayout* por lo que debe de tener un hijo donde se añadirá todo el contenido para desplazarse. Para el desplazamiento vertical se puede utilizar o un *ScrollView* o un *ListView*.

- **FrameLayout:** En Android, sirve para reservar un área de la pantalla para mostrar un único elemento. Sólo debe usarse para contener a una única vista ya que es difícil de organizar más vistas de manera que escale en diferentes tamaños de pantalla. Aun así se pueden añadir más vistas pero asignando a estas una constante denominada *layout_gravity*. Dicho atributo define como se debe colocar la vista tanto en el eje X como en el eje Y.

- **FragmentTabHost:** En Android, es un contenedor de pestañas y cada pestaña está asociada a un Fragment.

- **GridView:** En Android, es un ViewGroup que muestra los elementos en una cuadricula.

5.2 Librerías

5.2.1 Red Social

5.2.1.1 FACEBOOK

Facebook es una red social creada por Mark Zuckerberg mientras estudiaba en la universidad de Harvard. Se considera la red social más famosa del mundo y cuenta con más de 70 traducciones y un mil millones de usuarios activos.

Gracias a la API de Facebook para Android podemos realizar operaciones que se hacen en la versión web. En Vuqio concretamente utilizaremos esta API para poder registrar al usuario en la aplicación Vuqio con mayor facilidad. En el registro se le pedirá al usuario que comparta su lista de amigos y correo para poder identificar en Vuqio quienes son sus amigos. Además si el usuario opta por registrarse vía Facebook podrá compartir en su tablón de dicha red social sus gustos televisivos.

Para integrar Facebook a nuestra aplicación no solo hace falta integrarla a nuestro proyecto en *eclipse* sino también realizar varios procesos [15]. El primero de ellos es generar una clave hash en la terminal mediante la herramienta *keytool*. Una vez generada dicha clave habrá que registrar la aplicación en Facebook e introducir correctamente la clave generada anteriormente.

5.2.1.2 TWITTER4J

Antes de hablar de la API Twitter4j hay que decir que Twitter es otra de la redes sociales más importantes de todo el mundo. Fue creada por Jack Dorsey en marzo de 2006 y que tiene una gran importancia en Vuqio por varios motivos. Uno de ellos y el más importante es que 95% de las conversaciones sociales referentes a la televisión se producen en Twitter [16]. Gracias a este hecho la televisión está resurgiendo creando un concepto nuevo como es el de la segunda pantalla.

Gracias a la API Twitter4j podremos interactuar con Twitter en Vuqio. Twitter4j es una librería no oficial y gracias a ella se podrá realizar todos los requisitos citados en el apartado 3.1.

Para integrar esta API sólo basta con incluir en nuestro proyecto los .jar de Twitter4j [17]. Para permitir que el usuario pueda sincronizar su cuenta de Twitter en Vuqio hay que realizar un proceso de registro de la aplicación. Una vez completado el registro tenemos que obtener la *Consumer Key y Consumer Secret* para completar el proceso *oAuth.* OAuth es un protocolo de autorización segura que permite al usuario acceder a sus datos mientras sus credenciales están protegidas en todo momento. Una vez adquirida la *Consumer Key y Consumer Secret* son incorporadas en el proyecto para cuando el usuario se registre con sus credenciales, Twitter devolverá un token de autorización que la aplicación guardará y utilizará cuando el usuario realice operaciones, consultas, etc respecto a Twitter. Todas estas operaciones [18] son transparentes para el usuario que solo deberá poner su usuario y contraseña.

5.2.2 Peticiones Asíncronas

5.2.2.1 VOLLEY

Volley es una biblioteca de *Google* que se utiliza para hacer peticiones *http* de forma más óptima y sencilla. La librería hace llamadas asíncronas de forma transparente para el programador por lo que no hará falta utilizar *Asynctask.*

Gracias a Volley podemos priorizar y cachear peticiones con lo que una aplicación como Vuqio, que necesita hacer peticiones constantes, mejora en fluidez. Un caso crítico en el proyecto era poder consultar la guía de televisión de forma fluida. La guía de televisión contiene mucha información (toda la programación de tres días) y si el usuario empieza a consultarla de forma indefinida con *AsyncTask* sería muy difícil de controlar. Ya que habría peticiones que no son prioritarias y siempre tendría que realizar y esperar el resultado de la petición. En el caso de hacer una misma petición varias veces gracias a la caché que proporciona Volley, sólo se perderá tiempo en mostrar los datos en la interfaz.

Respecto a las consultas indefinidas que puede realizar el usuario en la guía de televisión Volley cancelará todas las peticiones anteriores.

Volley también puede ser utilizada para peticiones de imágenes, para ello proporciona un elemento propio en la construcción de una interfaz:

```
<com.android.volley.toolbox.NetworkImageView
    android:id="@+id/icon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="1dp"/>
```

Figura 5.1 Parte de la interfaz

Una vez creada la interfaz se introduce la url de la imagen que se quiere obtener (Figura 5.2):

```
NetworkImageView icon_tag = (NetworkImageView)
        findViewById(R.id.icon);
if (name_url != null) {
    icon_tag.setImageUrl(name_url, ImageCacheManager.getInstance()
            .getImageLoader());
}
```

Figura 5.2 Petición asíncrona de un imagen con Volley.

En la aplicación no se ha utilizado debido a que se utiliza otra librería para esto llamada *Picasso* (ver sección 5.2.2.2). Se introdujo esta librería ya que permite transformar la imágenes muy fácilmente, como por ejemplo en todas las interfaces donde aparecen imágenes circulares.

Volley ha sido útil en las peticiones de los *JSON* donde está contenida toda la información, por ejemplo en la parrilla televisiva. En la Figura 5.3 se muestra un ejemplo de uso.

```
    //Peticion JsonObjectRequest
    reqEpg = new JsonObjectRequest(Preferences.URL_EPG
            + "/" + fecha + "/" + time + ".json", null,
            new Response.Listener<JSONObject>() {
        @Override
        public void onResponse(JSONObject response) {
            try {

                //Respuesta obtenida => JSON
                VolleyLog.v("Response:%n %s", response.toString(4));
                JSONObject json = response;

                //Procesar los datos obtenido
                processJson(json, now);

            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            VolleyLog.e("Error: ", error.getMessage());
        }
    });
    // agregar la solicitud a la cola para ser ejecutado
    JsonController.getInstance().addToRequestQueue(reqEpg);
}
```

Figura 5.3 Petición asíncrona de un JSON con Volley.

Al comienzo del proyecto no se tenía constancia de dicha librería y era muy costoso realizar todas las peticiones con *Asynctask*.

5.2.2.2 PICASSO

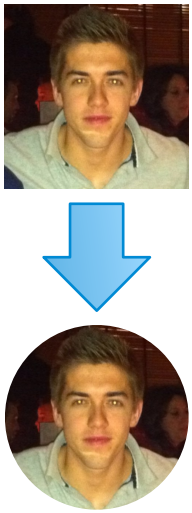
Picasso es una librería de Square que permite hacer peticiones prácticamente en una sola línea como se puede ver a continuación:

```
String url_image = currentProgram.getUrl_image_p();
if (url_image != null)
    Picasso.with(context).load(url_image).into(holder.imageView);
```

Figura 5.4 Petición de imagen con la librería Picasso.

Esta petición se realiza de forma asíncrona. Además todas las imágenes se almacenarán en caché. En este aspecto es importante para la aplicación de Vuqio ya que se realiza un gran número de peticiones de imágenes.

Para finalizar también hay que decir que permite transformar la imagen. En Vuqio por ejemplo se ha utilizado para mostrar todos los usuarios de Vuqio que están viendo un programa o los usuarios de Twitter que han comentado un programa. Esto es posible añadiendo una sentencia más como se puede ver en la Figura 5.1 que corta la imagen para que tenga un aspecto circular.



```
String url_image = user.getImage();
if (url_image != null) {
    Picasso.with(context).load(url_image)
    .placeholder(R.drawable.tagdefault)
    .transform(new CircleTransform())
    .into(holder.imageUser);
}
```

Figura 5.5 Transformación a una imagen circular con Picasso.

5.2.3 Interfaz

5.2.3.1 ACTION BAR SHERLOCK

"ActionBarSherlock es una extensión de la librería de compatibilidad diseñada para facilitar el uso del patrón de diseño action bar en una única API a través de todas las versiones de Android. La librería utilizará el action bar nativo cuando esté disponible o utilizará una implementación propia sobre vuestros layouts de una forma automática." [19]

La versión 3.0 de Android, Honeycomb, estaba orientada a las tablets y se introdujeron nuevos elementos como los *Fragments*. Los *Fragments* te permiten reutilizar código por lo que hace que la interfaz sea más modular. Debido a que Vuqio tenía como requisito estar disponible a partir de la versión 2.3 gracias a ActionBarSherlock ha sido posible.

Al comenzar el proyecto hizo falta incluir esta librería ya que no estaba en la librería de compatibilidad *android-support.*

5.2.3.2 Android-PullToRefresh

Android-PullToRefresh es una librería de Chris Banes y la podemos encontrar en *GitHub* [20].

Gracias a esta aplicación el usuario podrá actualizar la información que proporciona Vuqio, simplemente deslizando el dedo desde la parte superior del contenedor de la información hacia abajo (Figura 4.16 y Figura 4.17).

5.2.3.3 Pinned Section ListView

Pinned Section ListView es una librería de Halfbit y se encuentra en el repositorio *GitHub* [21] y el comportamiento de esta librería se puede ver en YouTube [22].

Se ha utilizado esta librería para organizar las alarmas del usuario. Las alarmas se agruparán por secciones de fecha de emisión. Y al recorrer las alarmas de un mismo día la cabecera de la sección se quedará fija mostrando siempre al usuario que día se emitirá el programa.

5.2.3.4 ViewBadger

ViewBadger es una librería de Jeff Gilfelt y se encuentra en el repositorio *GitHub* [23]. Se ha incorporado a Vuqio para notificar las conversaciones abiertas del usuario en la guía de televisión.

Se crea un objeto de la clase *BadgeView* en el método *onCreateOptionsMenu* sobrescrito (Figura 5.3).

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    //Carga el menu
    getSupportMenuInflater().inflate(R.menu.messages, menu);
    View icon_chat = null;
    RelativeLayout badgeLayout =  null;
    //Contenedor de la imagen
    badgeLayout = (RelativeLayout) menu.findItem(R.id.message_info).getActionView();
    if(badgeLayout != null){
        //obtención de la imagen para a continuación asociarla al BadgeView
        icon_chat = badgeLayout.findViewById(R.id.chat_info);
    }
    if(icon_chat != null){
        icon_chat.setOnClickListener(this);
        //Se incorpora al icono el BadgeView
        badge = new BadgeView(getApplicationContext(), icon_chat);

    }
    infoChats();

    return super.onCreateOptionsMenu(menu);
}
```

Figura 5.6 Creación del BadgeView.

Para actualizar el *BadgeView* de conversaciones abiertas, se hace una consulta a la base local y el resultado es añadido al objeto *BadgeView* como podemos ver en la Figura 5.5 a continuación:

```
public void infoChats(){
    if (badge != null) {
        int num_messages = new DatabaseAdapter(this)
        .getCountChatWhitoutRead();
        if (num_messages > 0) {
            badge.setText(String.valueOf(num_messages));
            badge.show();
        } else {
            badge.hide();
        }
    }

}
```

Figura 5.7 Actualización del objeto BadgeView.

5.2.4 Análisis de errores

5.2.4.1 HockeyApp

HockeyApp [24] se ha utilizado en la fase de desarrollo alpha y beta. Gracias a ella Vuqio ha sido distribuida a los testers. Hockey registra todos los errores producidos en la aplicación dando al detalle toda la información del error. Además permite a los tester dar feedback mejorando la calidad del producto pero no se ha incorporado dicha funcionalidad a Vuqio.

Para utilizar HockeyApp hace falta añadir al *Android-Manifest* de nuestra aplicación la Activity:

```
<activity android:name="net.hockeyapp.android.UpdateActivity" />
```

Si se desea hacer una distribución beta y notificar del error hay que añadir los permisos:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

En la actividad principal de la aplicación se ha añadido el envío de notificación errores, ya que cuando se produce un error de la aplicación no controlado esta se cierra:

```
@Override
public void onResume() {
    super.onResume();
    checkForCrashes();
    checkForUpdates();
}

private void checkForCrashes() {
    CrashManager.register(this, Preferences.APP_ID);
}

private void checkForUpdates() {
    UpdateManager.register(this, Preferences.APP_ID);
}
```

Figura 5.8 Comprobación de errores y actualizaciones de la Apk de Vuqio en HockeyApp.

La constante APP_ID se obtiene al subir la aplicación a HockeyApp ya que generará un token único para todas las versiones de la aplicación. Tanto la comprobación de errores como de actualizaciones de la aplicación se realizan de forma asíncrona.

5.2.5 Conexión

5.2.5.1 Autobahn

Autobahn [25] es una librería que implementa el protocolo *websocket* con el envío bidireccional de mensajes en tiempo real. Se ha implementado para la gestión del chat en Vuqio (Figura 5.9).

Establecimiento de la conexión y recibo de mensajes

```
try {
    mConnection.connect(websocketuri, new WebSocketHandler() {

        @Override
        public void onOpen() {
            //Conexion establecida
            Log.d(TAG, "Status: Connected to " + websocketuri);
        }

        @Override
        public void onTextMessage(String payload) {
            Log.d(TAG, "Got echo: " + payload);
        //Aqui se reciben los mensajes

        }

        @Override
        //Perdida de conexion
        public void onClose(int code, String reason) {
            Log.d(TAG, "Connection lost.");

        }
    });
} catch (WebSocketException e) {

    Log.d(TAG, e.toString());
}
}
```

Envio de mensajes

```
if (mConnection.isConnected()) {
    mConnection.sendTextMessage("{\"text\":\"" + texto + "\"}");
```

Figura 5.9 Gestión del Websocket.

5.2.6 Notificaciones

5.2.6.1 Google Cloud Messaging

Google Cloud Messaging [26] es otra librería de Google que permite enviar datos desde un servidor a un dispositivo con Android y enviar mensajes entre dispositivos Android. En Vuqio se ha utilizado para la gestión del chat. Un caso concreto es cuando el usuario no tiene la aplicación de Vuqio en ejecución o tiene el móvil apagado y otro usuario le escribe a través del chat de Vuqio. Si tiene el móvil apagado cuando encienda el móvil los servidores del Google detectan que se ha encendido el dispositivo y si tiene mensajes pendientes de recibir.

También se ha utilizado para actualizar las alarmas del usuario pero este no será notificado.

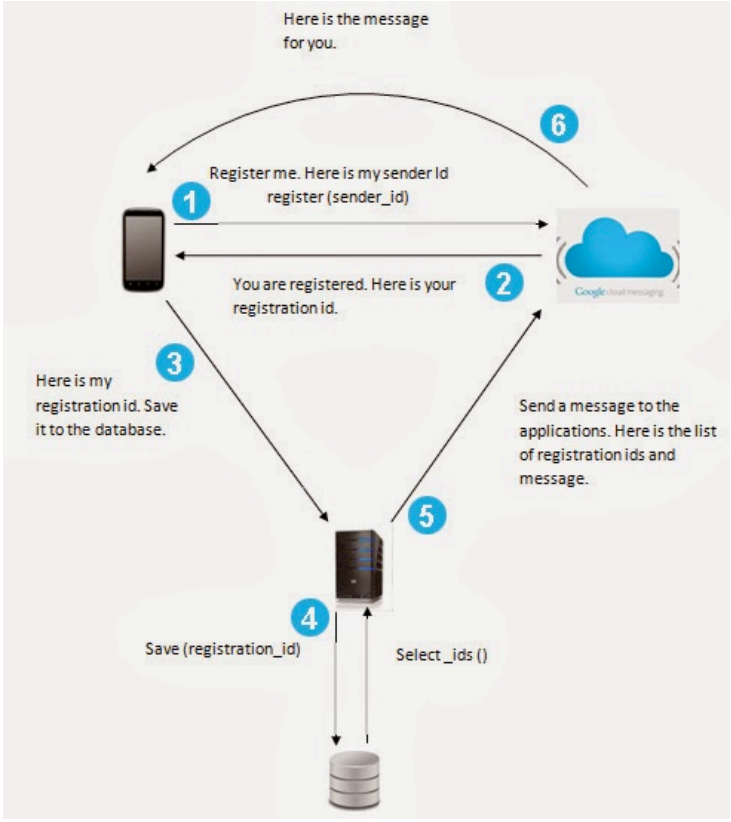


Figura 5.10 Ciclo de Vida de Google Cloud Messaging.

Como se puede ver en la Figura 5.10 cuando el usurio se registra en Vuqio a través de Twitter o Facebook correctamente a continuación se registra el dispositivo (paso 1) y el servicio de Google le responde con un identificador. Ese identificador es almacenado en la base de datos de Vuqio terminando el proceso de registro del usuario en Vuqio. Si el usuario enviara un mensaje a otro usuario sería enviado al servidor de Vuqio y este enviaría el mensaje al servicio de Google y este se encargaría de enviar los mensajes al otro dispositivo de forma transparente al desarrollador.

5.3 Interfaz

Debido a la gran complejidad de la aplicación, a continuación se va a detallar los bloques más importantes de Vuqio.

Antes de comenzar a explicar los bloques “Guia de Televisión” e “Información detallada del programa” Vuqio sigue el patrón MVC (Modelo Vista y Controlador). Y en relación a estos dos bloques la parte de controlador viene dada por este diagrama de clases (Figura 5.11).

MainActivity corresponde al bloque de la Guía de Televisión e InformationActivity a la Información detallada del programa. Las dos clases extienden de DrawerMenu ya que esta actividad genera el menú lateral disponible para las dos actividades hijas y gestiona los reemplazos de los *Fragments*.

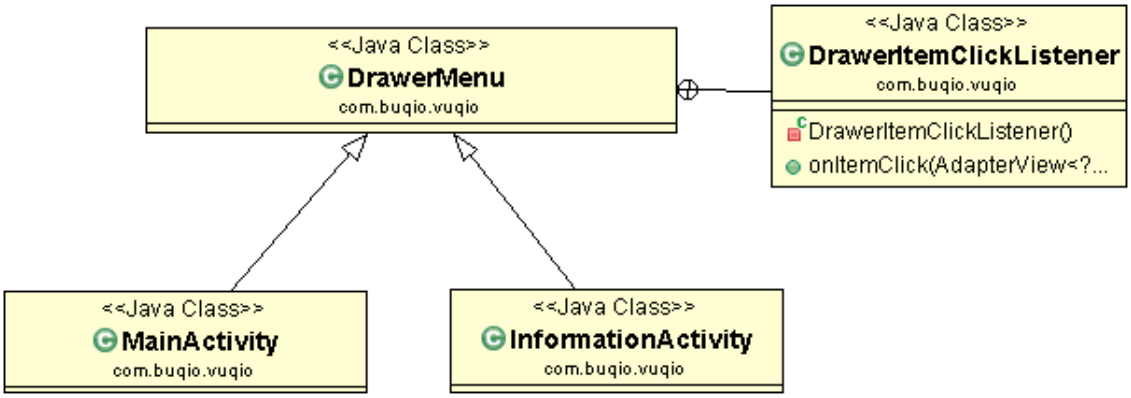


Figura 5.11 Diagrama de Clases Controlador.

5.3.1 Guía de televisión

En la Figura 5.12 vemos que la actividad MainActivity está compuesta por un *Fragment* que gestiona dos *Fragments* gracias a un *FragmentTabHost:*

- **Fragment de la parrilla televisiva:** Para mostrar la guía de televisión se ha utilizado un *ListView* con un adaptador customizado. Este adaptador por cada ítem utiliza un *ViewPager* con su respectivo adaptador personalizado. Esta solución es la mejor a corto plazo, pero en términos de rendimiento no. Esto es debido a que *ViewPager* está en una fase temprana de diseño y desarrollo, y se utiliza generalmente para gestionar *Fragments.* Como consecuencia no había fluidez en la visualización de la parrilla televisiva. Por lo que se creó una pila que almacena las vistas de los programas de todos los canales. Esta solución mejora la fluidez de la aplicación una vez que se han cargado dichas vistas.
  Para hacer consultas a la guía de televisión se ha desarrollado un *HorizontalScrollView* customizado. Cada hijo del *HorizontalScrollView* corresponde a un día de la semana. Cada día se divide en 48 intervalos para solo mostrar las horas en punto y medias horas. El desplazamiento que se haga sobre el *HorizontalScrollView* se calcula en que hijo está y cuanto se ha desplazado el cursor sobre el hijo para poder transformarlo a una hora concreta.

- **Fragment el chat global:** Para mostrar los usuarios de Vuqio como de usuarios que han comentado un programa en Twitter se ha utilizado un *GridView.* Este *Gridview* también tiene un *CustomAdapter.* Uno de los inconvenientes a la hora de desarrollar fue controlar si se había llegado al final del *GridView*, es decir, si el último elemento visible correspondía al ultimo usuario obtenido.
  La fluidez al visualizar los usuarios no está comprometida gracias a Picasso que se encarga de pedir las imágenes y su reciclado.

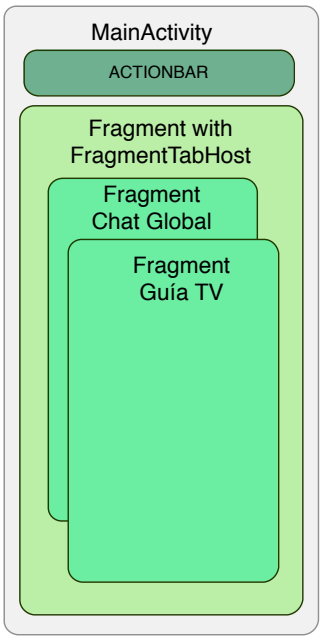


Figura 5.12 Estructura del Controlador de la interfaz principal.

5.3.2 Información detallada del programa

Este bloque, al igual que la parrilla de televisión, está compuesto por la actividad *InformationActivity* con un *Fragment* que contiene en un *FragmentTabHost* a otros 4 *Fragments* (Figura 5.13).

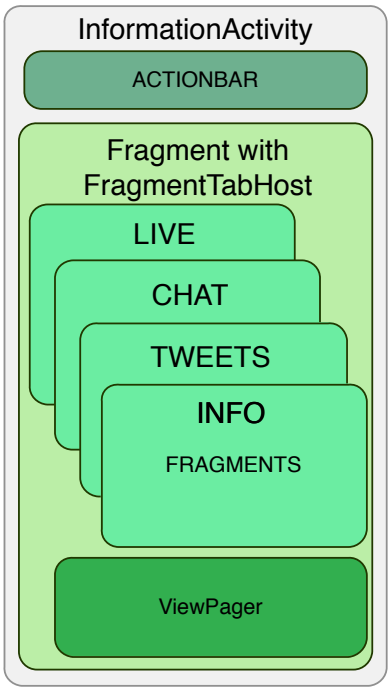


Figura 5.13 Estructura del Controlador de la interfaz información del programa.

Como se ha explicado antes, el *Fragment* que contiene y gestiona a los 4 *Fragments* también tiene a su cargo el carrusel de lo que se está hablando en el programa en directo. Dicho carrusel cada 5 segundos cambia de entidad como podemos ver a continuación:

```
public void autoPagesTags(final ViewPager listview, final int numElements){

    handler_tags = new Handler();

    runnable_tags = new Runnable() {

        public void run() {
            //Cada 5 seg se mustra la siguiente tag
            listview.setCurrentItem((listview.getCurrentItem()+1) % numElements, true);
        }
    };

    Timer swipeTimer = new Timer();
    swipeTimer.schedule(new TimerTask() {

        @Override
        public void run() {
            handler_tags.post(runnable_tags);
        }
    }, 5000, 5000);
}
```

Figura 5.13 Carrusel de entidades.

Este carrusel en la pestaña "LIVE" desaparece ya que es la misma información. Este detalle será controlado cuando se cambie de pestaña.

```
public void startRealTimeTag(){
    if(tThread == null){
        //Creamos el hilo
        tThread = new Thread(new Runnable() {
            public void run() {
                try{
                    //comprobamos que no ha sido interrumpido
                    while (!tThread.isInterrupted()) {
                        //realizamos peticion
                        download();
                        Thread.sleep(20000);
                    }
                }catch(InterruptedException ex){
                    Log.e("Tag","CAptura!!");
                }
                finally {
                    tThread = null;
                }
            }
        });
        //Empieza la ejecucuion del hilo
        tThread.start();
    }
}
```

Figura 5.14 Funciones de gestión del hilo.

La pestaña con más dificultad en cuanto a implementación es la pestaña "LIVE". En ella se crea un hilo que hace peticiones al servidor cada 20 segundos. Este hilo cancelará su ejecución cuando la aplicación permanezca en segundo plano, cuando el

usuario navegue a otro panel o cuando el usuario deslice el dedo desde la parte superior de la interfaz para realizar una petición manual. Para controlar el hilo se ha codificado dos sentencias, una que activa al hilo y otra que lo interrumpe (Figura 5.14).

5.4 Técnicas de desarrollo

5.4.1 Técnica ViewHolder

La Técnica ViewHolder surge debido a que cuando en un adaptador se van creando las vistas en el método *getView()* tenemos que utilizar una o varias veces la sentencia *findViewById()*. Esta sentencia es muy costosa y provocaba errores de memoria en la aplicación. Para resolver esto se incluyen los ID que identifican a cada elemento dentro de la propia *View*. El objeto *View* tiene un método *getTag()* y *setTag()*, que permiten guardar información en la vista y en este caso evitar llamadas innecesarias a *findViewById()*.Como se puede ver en la Figura 5.15.Y simplemente hace falta crearse una clase que en este caso se ha llamado *ViewHolder* y en el método *getView()* comprobar si la vista contiene los ID (Figura 5.16).

```
class ViewHolder{
    TextView name_tag;

}
```

Figura 5.15 Clase ViewHolder.

```
@Override
public View getView(final int position, View convertView, ViewGroup parent) {
    View item = convertView;
    ViewHolder holder = null;
    LayoutInflater inflater = (LayoutInflater) context
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    //Si la vista item no tiene informacion guardada --> findViewById
    if(item == null|| !(item.getTag() instanceof ViewHolder)) {

        holder = new ViewHolder();
        item = inflater.inflate(R.layout.tag_view,null , false);
        holder.name_tag = (TextView)item.findViewById(R.id.nameTag);
        item.setTag(holder);

    }
    //Contiene los ID
    else{

        holder = (ViewHolder) item.getTag();

    }
```

Figura 5.16 Utilización de la técnica ViewHolder.

Está técnica se ha utilizado en todos los contenedores de vistas de la aplicación mejorando el rendimiento de esta.

5.4.2 Triggers

Un trigger en una base de datos es un procedimiento que se ejecuta cuando se produce un cambio en la base de datos. En Vuqio se han utilizado para la gestión de las alarmas y se va a detallar a continuación:

- Uno de los triggers se ejecuta cuando se deja de seguir todas las emisiones de un programa y se eliminará el ID_FOLLOW de la tabla Follow y provocará borrado en cascada, eliminado las alarmas de la tabla Alarm asociadas al ID_FOLLOW y esto provocará la eliminación del programa asociado a la alarma (Figura 5.17).

```
String trigger_delete_follow_brand =
        "CREATE TRIGGER delete_follow_brand "
                +"BEFORE DELETE ON " + TABLE_FOLLOW_BRAND
                +" FOR EACH ROW BEGIN"
                +" DELETE FROM " + TABLE_ALARM + " WHERE " + FK_ALARM_IDFOLLOWBRAND + "= OLD." + ID_FOLLOW_BRAND + ";"
                +"END;";
String trigger_delete_alarm =
        "CREATE TRIGGER delete_alarm "
                +"BEFORE DELETE ON " + TABLE_ALARM
                +" FOR EACH ROW BEGIN"
                +" DELETE FROM " + TABLE_PROGRAM + " WHERE " + ID_BROADCAST + "= OLD." + TRACK_ID_BROADCAST + ";"
                +"END;";
```

Figura 5.17 Triggers de borrado en cascada.

- Debido a que la información de emisiones futuras enviada por el proveedor no son fiables provocan que haya que actualizar los programas guardados en la base de datos y por lo tanto hay que actualizar la tabla Alarm (Figura 5.18).

```
String trigger_update_alarm =
        "CREATE TRIGGER update_alarm" +
                "  AFTER UPDATE ON " + TABLE_PROGRAM + " FOR EACH ROW" +
                "  BEGIN " +
                "UPDATE " + TABLE_ALARM +
                "  SET " + TRACK_ID_BROADCAST + " = new." + ID_BROADCAST +
                "  WHERE " + TRACK_ID_BROADCAST + " = old." + ID_BROADCAST + ";" +
                "  END";
```

Figura 5.18 Trigger de actualización del campo ID_BROADCAST de la tabla Alarm.

6. Pruebas

6.1 Pruebas de usabilidad

Las pruebas que han tenido más importancia en Vuqio han sido de usabilidad ya que el objetivo principal de Vuqio era mejorar la experiencia del usuario. Dichas pruebas han sido recogidas gracias a la librería HockeyApp. Las pruebas se han llevado a cabo por diez personas registradas por Vuqio en HockeyApp y a las que se les ha suministrado versiones de la aplicación a lo largo del periodo de pruebas.

A continuación se muestra el registro de errores de la versión, que será subida a *Google Play Store,* del último mes junto al informe de errores:

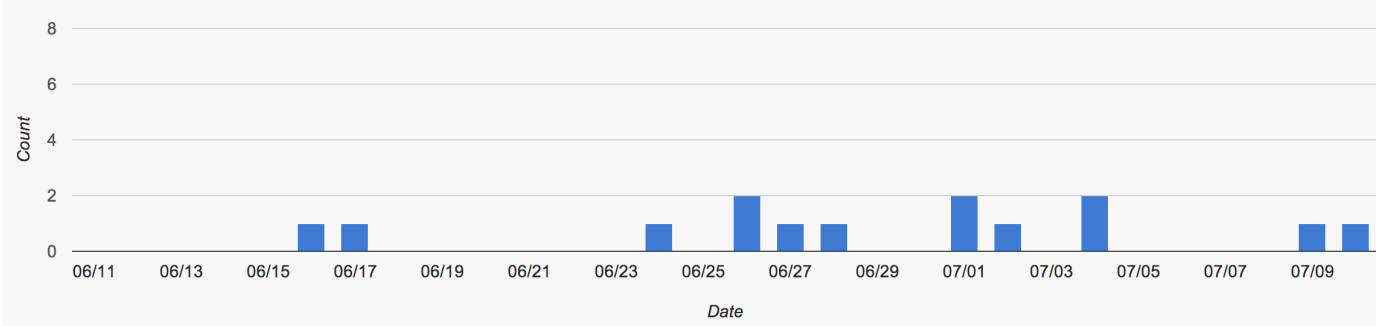


Figura 6.1 Gráfica de errores de Vuqio en el último mes.

4 open	**android.database.Observable.unregisterObserver** line 64 java.lang.IllegalArgumentException: The observer is null.	04 Jul 2014, 13:53 18
1 open	**controller.TwitterController$14.onResponse** line 634 java.lang.OutOfMemoryError	02 Jul 2014, 20:58 18
2 open	**adapters.horizontal_hours.CenterLockHorizontalScrollview.onScrollChanged** line 225 java.lang.NullPointerException	01 Jul 2014, 23:08 18
1 open	**fragments.FragmentInfo.onPause** line 241 java.lang.RuntimeException: Unable to pause activity {com.buqio.vuqio/com.buqio.vuqio.InformationActivity}: java.lang.NullPointerException	28 Jun 2014, 11:46 19
4 open	**VuqeaActivity.publishStoryInFacebook** line 365 java.lang.UnsupportedOperationException: Session: an attempt was made to request new permissions for a session that is not currently open.	27 Jun 2014, 13:32 ...
1 open	**android.support.v4.widget.ViewDragHelper.shouldInterceptTouchEvent** line 1004 java.lang.ArrayIndexOutOfBoundsException: length=1; index=1	26 Jun 2014, 23:25 18
1 open	**controller.TagController.loadTagsLinear** line 440 java.lang.NullPointerException	24 Jun 2014, 12:03 19
6 open	**VuqeaActivity.publishStoryInFacebook** line 365 java.lang.UnsupportedOperationException: Session: an attempt was made to request new permissions for a session that has a pending request.	17 Jun 2014, 00:01 ...
3 open	**adapters.ListSocialAdapter.getView** line 31 android.view.InflateException: Binary XML file line #27: Error inflating class com.facebook.widget.LoginButton	10 Jun 2014, 21:59 ...
1 open	**verticalhorizontallistview.MyProgramPagerAdapter.getProgram** line 368 java.lang.IndexOutOfBoundsException: Invalid index 2, size is 2	09 Jun 2014, 15:13 18
1 open	**android.app.ActivityThread.performDestroyActivity** line 3661 java.lang.RuntimeException: Unable to destroy activity {com.buqio.vuqio/com.buqio.vuqio.MainActivity}: java.lang.IllegalArgumentException: The observer is null.	07 Jun 2014, 15:52 18

Figura 6.2 Informe de errores de Vuqio en el último mes.

Para poder resolver los errores Hockey guarda el informe de error completo detallándote la línea del código donde se produjo el error (Figura 6.3).

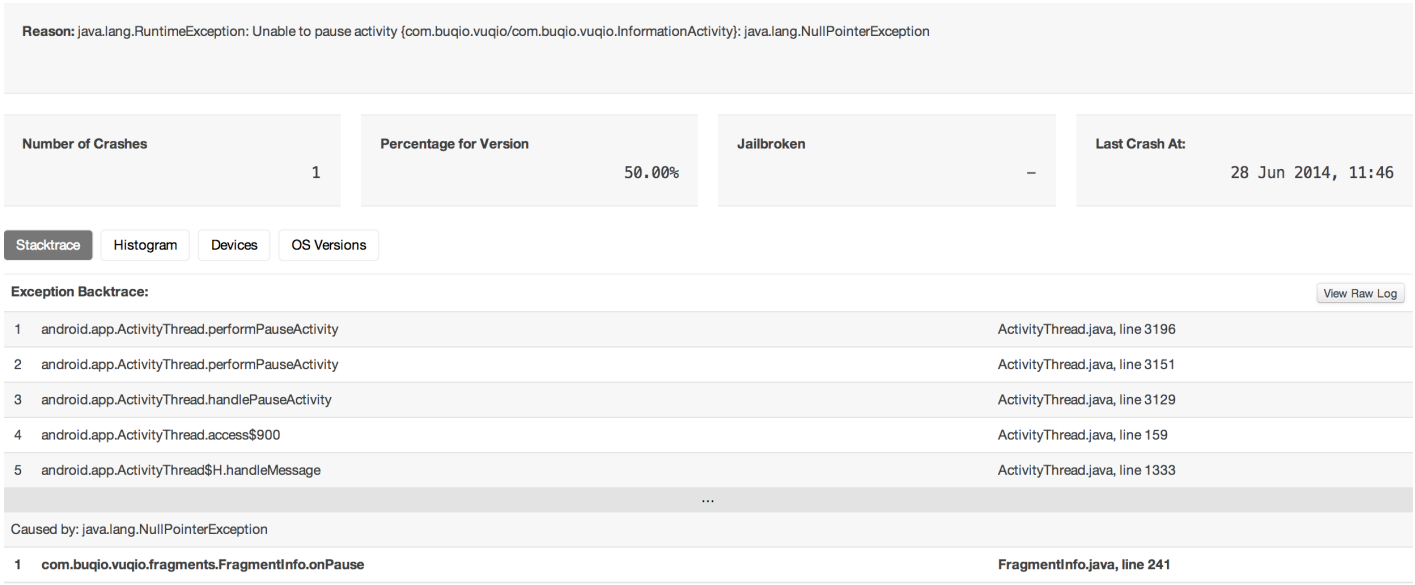


Figura 6.3 Traza de un error de Vuqio en el último mes.

6.2 Pruebas de sistema

Debido a que Vuqio soporta versiones a partir de la 2.3 también se han realizado pruebas de sistema. Para poder controlar las versiones del sistema operativo Android por debajo de la 3.X en muchos casos se ha añadido un control de versiones en el código (Figura 6.4).

```
public static void generateNotification(Context context, String title,
        String subtitle) {

    int currentapiVersion = android.os.Build.VERSION.SDK_INT;

    if (currentapiVersion >= android.os.Build.VERSION_CODES.HONEYCOMB_MR2) {
        newVersionNofity(context, title, subtitle);
    } else {

        oldVersionNofity(context, title, subtitle);
    }

}
```

Figura 6.4 Control de versiones.

Prueba 1: Funcionamiento estándar 2.3.6

Se ha probado la aplicación con un *Samsung Galaxy Young*.

Prueba 2: Funcionamiento estándar 4.1.2

Se ha probado la aplicación con un *Samsung Galaxy S2*.

Prueba 3: Funcionamiento estándar 4.3

Se ha probado la aplicación en un *Samsung Galaxy S3 y S4* cada una con distintas densidades xhdpi y xxhdpi respectivamente.

Prueba 4: Funcionamiento estándar 4.4.2

Se ha probado la aplicación con un *Samsung Note 3 y Nexus 5*.

Prueba 5: Funcionamiento estándar 4.4.3

Se ha probado la aplicación con un *HTC One*.

7. Análisis

Google Analytics es una libreía de Google para recoger información de interacción por parte del usuario y así poder mejorar la experiencia de los usuarios en Vuqio. A continuación se mostrará algunas estadísticas relacionadas con Vuqio.

En la Figura 7.1 se muestra el número de usuarios que se descargaron la aplicación y los usuarios activos. Como se puede ver en Mayo se produjo un gran número de descargas gracias a la publicación de una noticia en uno de los más importantes blogs de desarrolladores [27].

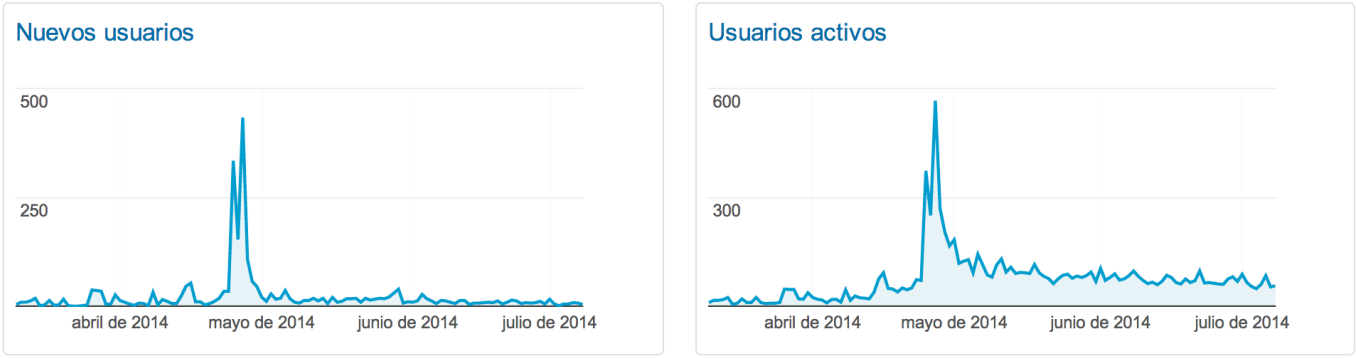


Figura 7.1 Usuarios en Vuqio por Google Analytics.

Figura 7.2 Información general de sesiones de Vuqio.

Gracias a Google Analytics podemos saber en que zona geográfica se producen más sesiones por parte de los usuarios y los dispositivos más utilizados (Figura 7.2).

Como se ve en la Figura 7.3 las pantallas más importantes de Vuqio son la guía de televisión y la información detallada del programa dividida por las cuatro pestañas descritas en el apartado 4.1.2.

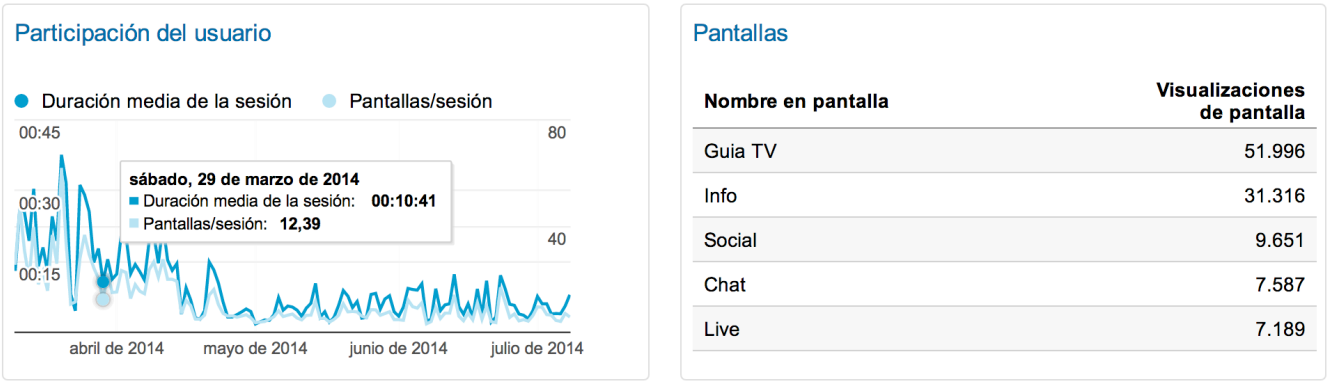


Figura 7.3 Participación del usuario y pantallas más visitadas.

```
@Override
public void onStart() {
    super.onStart();
    EasyTracker easyTracker = EasyTracker.getInstance(context);
    easyTracker.set(Fields.SCREEN_NAME, "Info");
    easyTracker.send(MapBuilder.createAppView().build());
}

@Override
public void onStop() {
    super.onStop();
    EasyTracker.getInstance(context).activityStop((Activity) context);
}
```

Figura 7.4 Envío de información a Google Analytics de inicio y finalización de sesión en la pantalla Info de Vuqio.

Estas estadísticas son posibles gracias a la incorporación de la librería en el proyecto. Simplemente hace falta añadir las siguientes líneas de la Figura 7.4 en los métodos *onStart* y *onStop* del ciclo de vida de los *Fragments.*

8. Conclusiones

En este proyecto se ha desarrollado una aplicación Android estable para la empresa Buqio S.L la cual permite al usuario interactuar con otros usuarios mientras ve la televisión.

La aplicación desarrollada permite descubrir nuevos contenidos gracias al sistema de entidades y ver la opinión de otros usuarios en Twitter de sus programas favoritos. Unifica los servicios de Whatsapp, Twitter y Facebook en una sola aplicación permitiendo al usuario una mayor interacción.

Vuqio dota de mayor nivel de interactividad a los programas televisivos. Esto, por un lado, provee a las marcas con un modo de posicionamiento publicitario y, por otro, supone una nueva forma de ver la televisión, más divertida, social, interactiva y participativa.

Vuqio sigue creciendo en funcionalidades, como que los usuarios puedan ver la televisión en su dispositivo móvil. La importancia de series online ha hecho que Vuqio permita en un futuro poder crear foros relacionados con series, actores, actrices, etc y que los usuarios se suscriban estando informados de todos los acontecimientos publicados en este. Otra funcionalidad estudiada ha sido que los usuarios puedan interactuar con un programa gracias a encuestas y juegos haciendo una televisión más divertida.

REFERENCIAS

[1] http://ecommerce-news.es/mobile/apps/nace-vuqio-nueva-aplicacion-para-enriquecer-la-experiencia-de-ver-la-tv-16251.html
[2] http://developer.android.com/google/gcm/gcm.html
[3] http://www.androidhive.info/2012/10/android-push-notifications-using-google-cloud-messaging-gcm-php-and-mysql/
[4] http://www.bbc.co.uk/ontologies/po
[5] http://wiki.nginx.org/Main
[6] http://akka.io/
[7] http://redis.io/
[8] http://www.mongodb.org/
[9] http://www.w3.org/XML/
[10] http://rubyonrails.org/
[11] http://www.json.org/
[12] http://aws.amazon.com/es/s3/
[13] https://dev.twitter.com/docs/api/streaming
[14] http://developer.android.com/sdk/index.html
[15] https://developers.facebook.com/docs/android/getting-started/
[16] https://business.twitter.com/es/twitter-tv
[17] http://twitter4j.org/en/index.html#download
[18] https://blog.twitter.com/2013/login-verification-on-twitter-for-iphone-and-android
[19] http://actionbarsherlock.com/
[20] https://github.com/chrisbanes/Android-PullToRefresh
[21] https://github.com/beworker/pinned-section-listview
[22] https://www.youtube.com/watch?v=mI3DpuoIIhQ
[23] https://github.com/jgilfelt/android-viewbadger
[24] http://hockeyapp.net/features/
[25] http://autobahn.ws/android/
[26] http://developer.android.com/google/gcm/index.html
[27] http://www.genbeta.com/deskmod/vuqio-centraliza-toda-la-actividad-social-de-un-programa-de-television-en-tu-movil